TECHNIQUES FOR OPTIMIZING TIME-STEPPED SIMULATIONS

Karthik Shenoy

B.E (Computer Science), Mangalore University

A Thesis Submitted

for the Degree of Master Of Science

National University of Singapore

July 2003

Acknowledgements

I wish to sincerely thank Dr. Gary Tan and Dr. Tay Seng Chuan for having the vision to guide the development of the techniques outlined here.

I must also thank the entire research community that is actively involved in the field of distributed simulations for the often extensive excerpts I have taken from their material.

I owe a debt of gratitude to others who have helped make this work possible - members of the Computer Systems Laboratory and my friends here at the university.

The main bulk of the experimental work was carried out at the Scientific Computation and Multimedia Communication laboratory at the Physics Department in the National University of Singapore. I also want to express my gratitude to the lab technologists there including Madam Tay Bee Hwee and Mr. Lim Hwa Ngee for going that extra-mile sometimes.

To Mr. Thio Seng Joo at Defence Science and Technology Agency and Dr. Come Raczy at the School of Computing who have always enriched me with their tips, I am much obliged. A special note of thank you to Prof. Rassul Ayani for his constructive feedback.

For the incessant support during the course of my work, my heartfelt gratitude to my family.

Contents

List of Figures

List of Tables

Abstract

In this thesis, we propose some optimizations for reducing global synchronization in traditional time-stepped simulations. The work is mainly focused on interactive type of simulations. Traditional time-stepped simulations are known to be efficient when simulation events are both frequent and dense. However, when simulation events are less frequent (when compared to the size of time-steps) the performance of time-stepped simulations degrades noticeably. This work aims to improve the performance of traditional time-stepped simulations when the frequency of events is low and to maintain the efficiency of time-stepped simulations when the frequency is high. For simulations with tight real-time interactive constraints, the optimization is achieved by maintaining information about future events at the host. In cases where lookahead information is available and the real-time constraints are relaxed a barrier synchronization based simulation is used.

In the course of any simulation it is possible that simulation parameters such as event density, lookahead, real-time constraints keep varying. The research aims to achieve simulation efficiency by switching between the optimized simulation techniques (traditional, piggyback-based and lookahead-based) and to achieve this switching we introduce a concept called 'super-stepping'. A probabilistic method is used to estimate suitable 'super-step' sizes.

Chapter 1

Introduction

Computer Simulations are widely employed by the scientific community today for studying, analyzing and predicting the behaviors of real-world systems. The physical world that we live in is inherently parallel [Nic88] and this means that parallel and distributed models will be capable of simulating these physical systems more realistically. In addition, when properly implemented, parallel and distributed simulations surpass sequential simulations [Tay98] in terms of execution time, tolerance to faults [Mis86] and flexibility. Further, when high complexity is involved analytical modelling can prove to be intractable. Direct measurement, where it is possible, offers high precision but is usually even more time consuming and costly [Jai91], so simulation represents a middle-way alternative. Hence, the parallel and distributed simulation technology currently has been gaining widespread acceptance [Fuj00] in areas such as defense applications (war gaming simulations, training emulators), high performance computing (evaluation of telecommunication networks [THFD98], jet propulsion dynamics) and entertainment.

Most physical systems change with the passage of time [JBN96]. Hence any simulation that emulates a physical system must have some representation of the time attribute. In addition, the simulation must provide a representation for the current state of the system and a mechanism to change this current state to model the evolution over time. In computer simulations the state information is represented using state variables and an abstraction of time is used to represent the physical

time [Lam78]. This abstraction, commonly known as simulation time is usually not equal to the real physical time in terms of units and progression intervals, and this important concept is discussed next.

The following definitions [Fuj00] will be used throughout this thesis:

1. Physical time: This refers to the time in the physical system being simulated.
2. Simulation time: The abstraction of time used by the simulation to represent physical time.
3. Wall-clock time: Time in the real world during the execution of the simulation.

These notions of time will be amply clear from the following example. Consider a simulation of the queue on a particular working day at a local bank. The physical time extends from say 9:00 AM to 12:00 Noon on Monday, October 14, 2002. Simulation time might be represented by a number with each unit representing one second. If the computer simulation is completed in 10 minutes from 4:00 PM to 4:10 PM on Friday, October 11, 2002 then the time during which the computer simulation was running is the wall-clock time.

1.1 Classification of Simulations

Simulations can be classified based on three important underlying aspects:

1. Method of adherence to causality constraints
2. Range of values assumed by state variables
3. Mechanism of state change with respect to time

1.1.1 Method of Adherence to Causality Constraints

Simulations are often classified based on the manner in which causality constraints are enforced. Conservative approaches [CM79, Lub89a] strictly avoid the possibility of any causality error ever occurring. All conservative approaches rely on some

strategy to determine when a particular event is safe to be processed. Thus, an event with time-stamp t is scheduled for execution only after it can be concluded with certainty that all events with time-stamp less than t have already been processed. We will further discuss conservative approaches in Section 2.2.

Optimistic approaches [Jef85] on the other hand, do not strictly adhere to the local causality constraint. It proceeds greedily, assuming that events in its incoming queue are safe and then uses a detection and recovery approach to resolve the causality errors. Recovery is performed either by rolling back state variables or by sending a negative message (anti-message) to nullify the effect of a previously sent message detected to be out of order. This work is not dealing with any of the optimistic simulation approaches and therefore we will not be discussing this further.

1.1.2 Range of Values Assumed by State Variables

Physical systems can be classified into discrete or continuous, depending on their behavior with respect to the passage of physical time. A discrete system [JBN96] is one in which the state variable(s) changes only at a discrete set of points in time. For example, a classroom with the state variable defined as the number of students in the class, can be a discrete system for simulation. On the other hand, a continuous system is one in which the state variable(s) changes continuously over time. An example is the water level in a reservoir which keeps changing continuously due to rainfall, consumption for human use, evaporation etc.

The values assumed by discrete state variables versus those assumed by continuous state variables against simulation time are shown in Figure 1.1.

It follows that in cases where the system state changes all the time, not just at the time of some discrete event, continuous simulation [LK91] is appropriate. In this kind of simulation, the model is often a set of differential equations solved with numerical methods where time is one of the free variables [Eks00]. It is worth noting that time-stepped discrete event simulations are sometimes used to simulate

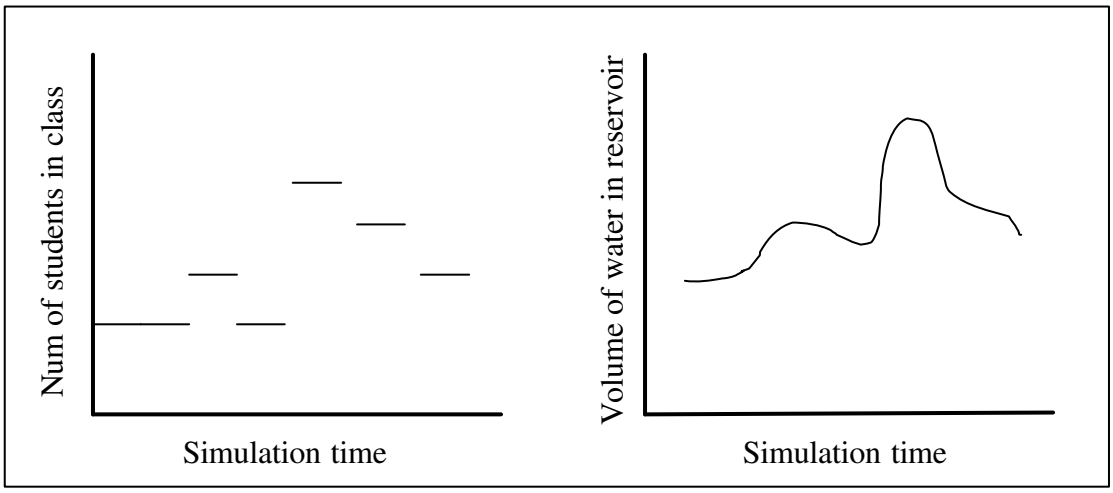

Figure 1.1: Continuous and Discrete Systems

continuous systems because they are easier to implement and they are capable of providing close approximations to continuous systems.

1.1.3 Mechanism of State Change with respect to Time

Based on the mechanism on which state changes take place, simulations are broadly categorized into time-stepped and event-driven. In time-stepped simulations, all participating entities in the simulation are at the same time-step at any point in wall-clock time.

Typically, the entire span of simulation time is divided into equal sized time-steps and simulation advances from one time-step to the next. At the i^{th} step, the algorithm simulates all events that fall in the time interval $[(i-1)\Delta, i\Delta]$, where Δ is a design parameter [EGLW93]. If Δ is very small the efficiency of the method degenerates since much coordination effort is wasted on intervals containing no events. Thus, Δ is chosen large enough so that a Processing Element (PE) typically has several events to process in any given interval $[(i-1)\Delta, i\Delta]$ but the simulation becomes coarse-grained because all events in the interval are simulated as if they occurred simultaneously.

Contrarily, event-driven simulations advance based on time-stamped events. Due to this reason it is not necessary for all participating PEs to be at the same logical

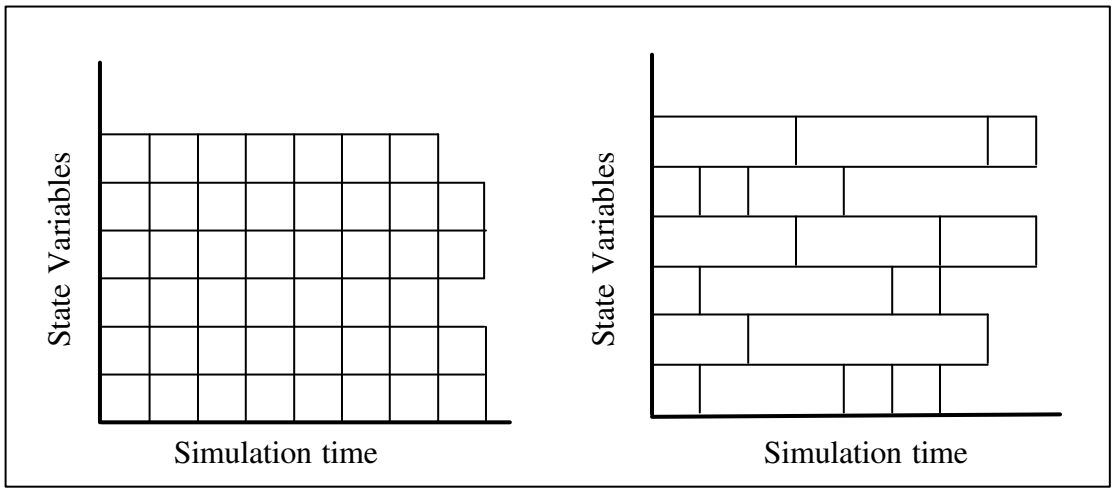

Figure 1.2: Time-Stepped versus Event Driven Simulation

time. Since discrete event simulations allow non-uniform advances in time they are usually more efficient [EGLW93]. However, time-stepped simulation is the preferred choice when the event coverage is dense and when state updates of participating entities need to be performed or need to be visible at regular intervals of time.

1.2 Real-time Interactive Simulations (RTIS)

Real-time interactive simulations play an active role in training based simulations of today. Many military and defense related simulations belong to this category of simulations [BD97]. These simulations are a cross-breed of two categories of simulations, namely:

- Interactive Simulation
- Real-Time Simulation

1.2.1 Interactive Simulation

In the early days of computer simulation, interactive simulation was a term much favored by analog computer manufacturers wishing to emphasize the advantages of their products over digital computers. Analog computers do in fact, provide a commendable degree of user interaction, especially at run-time, and offer many lessons for the designers of interactive digital simulation systems. With digital systems, on

the other hand, the development of interactive simulation facilities has taken many years to gather momentum. With the increase in the available processing power, the prospects for interactive simulation have been dramatically changed. Indeed, computer interactive simulation is currently undergoing an explosion of innovation and a dramatic increase in its range of applications, that today no one would think of realizing any complex system (an aircraft, a defense strategy, training of complex systems in science and engineering, entertainment, emergency planning to prepare for earthquakes and other disasters) without creating its simulation counterpart first. But simulation has been known to be notoriously expensive. As a result of this, distributed interactive simulation, in which the environment could be distributed both in terms of memory and geographical locations, appears to be a promising solution.

Distributed Interactive Simulation (DIS) provides an infrastructure to build large-scale simulations for the simulation of highly interactive activities by interconnecting several types of simulators via a network. This new technology has brought a new set of issues and challenging problems to solve. The new tools of DIS are able to support substantially the solution of highly complex problems from mathematical modelling in physics, engineering, and biology. This is an important direction of DIS technology applications.

As we look towards the future, an important direction of research in distributed interactive simulation appears to lie in the design of tools and environment that facilitate the transfer of distributed interactive simulation to the general simulation community, while extending their use in new real-time application domains.

1.2.2 Real-Time Simulation

Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation but also on the time at which the results are produced. The real-time simulation covers a range of topics. It involves the communication of values of system variables between those parts of the system which are modelled on the computer and those parts which are external to

the computer. The external equipment may include sub-systems modelled on other computers. The concept of communication interval, as used in many simulation languages, can be extended to cover the process of communication between different parts of such systems.

The major area of application of real-time simulation techniques has been in the design of training simulators. The most time critical of these applications occur in the aerospace field, particularly in military applications. In many of these applications, the integration process represents quite a small percentage of total execution time much of which is devoted to multiple variables function generation. Hence, there is a need to tackle these problems, in any attempt to include real-time features in a general interactive simulation facility.

Indeed, real-time simulation is necessarily interactive: if only to represent the output of the model directly to the user in a more realistic way. The incorporation of actual plan or human operator into a simulation usually calls for interactive real-time simulation. This process is most highly developed in virtual reality simulators, training simulators used to train civil and military aircraft pilots, power plant and process plant operators in which the computer simulation accepts operator inputs from instruments and other indicators on a realistic control panel. In many cases, the computer also produces realistic visual, sound and motion effects to heighten the illusion of reality.

The increasing computational complexity of simulating such systems has led researchers to conduct simulation studies on such platforms as multi-computer network architectures. In spite of considerable advances in recent years in the development of distributed simulation and interactive computing resources, there is undoubtedly a need for new tools of DIS that would be able to support substantially the solution of highly complex problems of mathematical modelling in physics, engineering,and biology.

1.3 Objective of Research

Though a lot of research work has gone into optimizing event-driven simulations relatively less work has been done in the area of real-time interactive time-stepped simulations. This is mainly due to the fact that time-stepped simulations are practical only in situations with high event density. Moreover, achieving optimal parallelism becomes harder when time-stepping constraint is imposed, because synchronizations at each time-step need to be mandatorily performed even though achieving of lower simulation overheads may demand otherwise.

However there are some favorable properties of time-stepped simulations that have resulted in its widespread use in the defense community. Time-stepped simulations tend to be extremely efficient when the event stream is dense in nature. By dense, we mean that the inter-event time is extremely low. Event driven simulations due to their high synchronization overhead tend to be slower than time-stepped simulations for such cases. However, this strength of time-stepped simulations is also its greatest weakness because when events are rare the time-stepping method becomes highly inefficient compared to the event-driven simulation method.

Time-stepped simulations also possess the ability to display to the user the entire state of the simulation at regular points in simulation time during the course of the simulation. Event driven simulations have the capability of displaying simulation states only at various points in wall-clock time by using barrier synchronization mechanisms.

In the real world the density of events can be highly varying depending on the period of time being simulated. Defense simulations have marked differences in event densities during time of war (high event density combined with need for fine-granularity and real-time response) as compared to peace time (relatively low density and response time is not mission critical). This research work is aimed at developing synchronization algorithms that adapt well to both extremes.

During intervals of time in which event density is low (sparse regions) each entity invests processing time in scanning through the event queue at each time-step even if the time interval does not contain events. In traditional time-stepping participating entities are informed about each time-step regardless of whether they have events to be processed or not. Our proposed piggy-backing technique informs participating PEs about a time-step only when there are events to be processed at that time-step. Due to the reduced synchronization overhead the technique is suitable for simulations which operate under strict real-time constraints.

The lookahead parameter of participating PEs is usually not explored in time-stepped simulations. The reason being, time-stepped simulations have a synchronization happening at every time-step and exploring lookahead information does not lead to performance improvements. If lookahead information is available, then performance improvements can be achieved by making use of them. However, when lookaheads are used to achieve simulation efficiency the global update of simulation state can be performed only at barriers. Thus, strict real-time constraints may not be met.

It is important that the simulation switches between the various techniques mentioned above, depending on the prevailing parameters such as event density, real-time criticality etc. To this effect, a simple super-stepping technique is developed. While a larger super-step size reduces the step-size calculation overhead, it decreases the chances of performing global saves and of performing a simulation strategy switch. On the other hand, a smaller super-step size increases the amount of overhead incurred. As simulation is a dynamic process, an adaptive approach based on the progress of simulation seems appropriate. In this thesis, we abstract the number of future events by a probabilistic approach. As we assume that the events occurring in non-overlapping super-step intervals are independent, the Poisson probability density function is used in our estimation when a chi-square measure qualifies the Poisson fit. Otherwise, we use a simple linear fit to estimate the next super-step size. The host receives the estimates of next super-step size from each of the participating

entities and a mean of these estimated values is used.

1.4 Overview of Thesis

The rest of this thesis is organized as follows. Chapter 2 discusses the problem of synchronization in distributed simulations. A survey of the previous work done in the area of conservative synchronization is then presented. Chapter 3 describes optimizations using piggy-backing and using lookaheads. These optimizations aim to reduce the number of overhead messages and the overhead time in time-stepped simulations. As simulation is a dynamic process, it is possible that simulation parameters such as number of participating PEs, lookahead of these PEs and cost of inter-PE communication keep varying. Therefore, optimizations that yield good results during some intervals of time may not perform well in others. Chapter 4 explains our super-stepping approach that will help us switch between simulation techniques depending on prevailing simulation parameters. In Chapter 5, the war-game simulation model used in the experimental work is explained. The performance of our optimization in terms of overhead messages, synchronization steps and overhead time is also presented. Chapter 6 concludes this thesis and outlines future research directions.

Chapter 2

Synchronization - Techniques and Related Work

In DES all events have an associated time-stamp indicating at what simulated time the event should occur. In a sequential simulation the events are processed in time-stamp order to ensure that an event cannot affect its past history (if this should occur it is known as a causality error). In Parallel Discrete Event Simulation (PDES), each PE asynchronously processes events in parallel with the other PEs. Thus, instead of a centralized clock, the PE has its own logical clock, the Local Virtual Time (LVT), that runs independently of the other PEs. For this reason, the PEs have to be synchronized to avoid causality errors. In this chapter we discuss some previously proposed synchronization techniques and their pros and cons.

2.1 The Synchronization Problem

Synchronization techniques impose the Local Causality Constraint (LCC) [Fuj00] to ensure that the simulation remains free of causality errors. The LCC states:

A discrete-event simulation, consisting of PEs that interact exclusively by exchanging time-stamped messages obeys the local causality constraint if and only if each PE processes events in non-decreasing time-stamp order.

As discussed in the previous section simulation techniques that strictly adhere to this constraint are known as conservative synchronization algorithms [Fuj93]. A brief survey of the work done in this area is presented next.

2.2 Survey of Previous and Related Work

Conservative synchronization algorithms fall under 3 major categories (see Figure 2.1) depending on the approach they take towards handling deadlock. The first category of algorithms aims to avoid deadlocks entirely. The second category allows simulations to proceed until a deadlock is detected and then deadlock recovery algorithms [CM81] are applied to resume simulation progress.

The third category is similar to the second category but the only difference is that these algorithms explicitly stop the progress of the simulation at various points in simulation time rather than relying on the system becoming deadlocked. In order to achieve this, these algorithms rely on a mechanism called barrier synchronization.

Many implementations of conservative simulation adopt the null message approach, originally developed by Chandy, Misra and Bryant (CMB) [CM79] for the synchronization of simulation events among PEs in parallel simulation. However, the CMB protocol is rather systematic, because each PE at the end of a simulation pass, sends null messages to all PEs, even if it is not necessary to do so. The amount of null messages required in a simulation can be extremely high even for a relatively small problem size.

Many enhancements have been proposed [FN92] to the original CMB approach. The Cai-Turner approach [CT90] significantly reduces null message traffic in simulations whose inter-node communication graphs contain cycles. The carrier-null messages used by the approach carried additional information such as the transit history of the nodes it has previously visited. This information was used by the individual nodes before advancing their simulation time. However, the Cai-Turner scheme can fail to be effective in the case of certain communication graphs such as

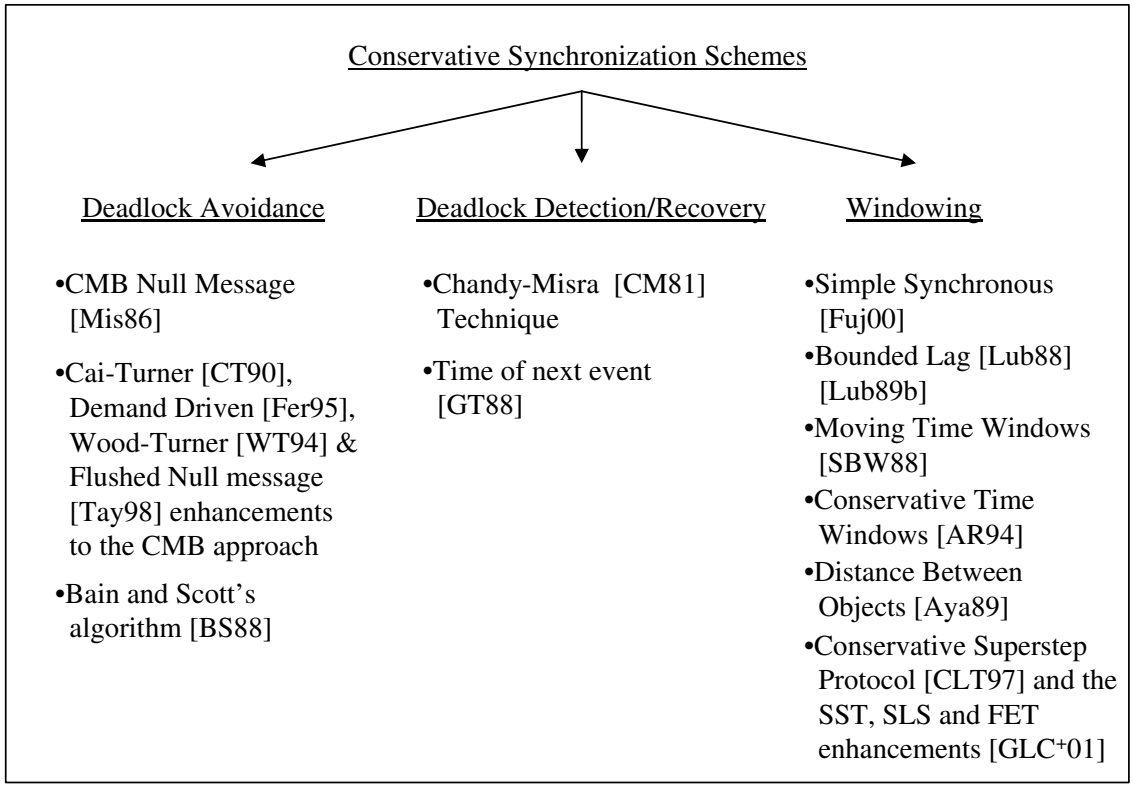

Figure 2.1: Conservative Synchronization Algorithms

those with nested cycles. The Wood-Turner carrier-null scheme [WT94] again aims to reduce the null message overhead caused by the existence of cyclic topologies in parallel simulation. The flushing mechanism reduces the growth of null messages by preventing a PE from sending null messages with the same time-stamp value to other PEs. When a PE schedules new null message in its output channel, all null messages with time-stamp value smaller than the new one are flushed [Mis86]. The flushed null message technique [Tay98] avoids deadlocks by collating all the incoming null messages from the incoming channels and flushing out one copy with the largest time-stamp. Such a flushing mechanism ensures that the PE topology has at least one null message in transit, thus avoiding any chances of deadlock. Fujimoto proposes an alternative approach to the conventional deadlock avoidance mechanism, by sending null messages on demand-driven basis [BS88, Fer95]. A PE sends out a null message when it receives a null message request from another PE.

Groselj and Tropper proposed the time-of-next-event algorithm [GT88] that deals with the case where more than one PE is mapped onto the same physical processor.

It uses the greatest lower bound of the time-stamps of the event messages expected to arrive next at all the empty links on the PEs on that processor to unblock the PEs on that processor. However, this algorithm has its drawback in the fact that it cannot detect deadlocks that span across processors.

Windowing algorithms are an important class of synchronization algorithms for parallel discrete event simulation. Lubachevsky's bounded lag algorithm [Lub88, Lub89b] was the first window-based algorithm and was followed by Ayani's Distance between objects approach [Aya89]. Windowing algorithms proceed in three distinct phases, each separated by barrier synchronization [DJ91]. In the first phase, PEs determine the simulation window cooperatively. The floor of the window is the minimum time-stamp in the system. The ceiling of the window is chosen such that all events with time-stamps falling within the window can be executed concurrently without the possibility of any causality errors. Using the terminology of Chandy and Sherman [CS89] events falling within the window are unconditional events. That is, their execution cannot be affected by any other event in the system. Events with time-stamps outside the window boundary are conditional events and their computation may be affected by some other events in the system.

The second phase of a windowing algorithm consists of the concurrent execution of all events having time-stamps within the window. In the third phase, events generated as a result of the processing in the second phase are passed on to other PEs. The primary difference among the various windowing algorithms is the mechanism to determine which events can be processed concurrently without causing causality errors.

Lubachevsky uses a moving simulated time window to reduce the overhead associated with determining when it is safe to process an event. The lower edge of the window is defined as the minimum time-stamp of any unprocessed event. Only those unprocessed events with time-stamps within the window are eligible for processing. The purpose of the window is to reduce the search space one must traverse in determining if an event is safe to process. For example, if the window extends from

simulated time 10 to time 20, and the application is such that each event processed by a PE generates a new event with a minimum time-stamp increment of 8 units of simulated time, then each PE only needs to examine the unprocessed events in neighboring PEs to determine which events are safe to process. No unprocessed event greater than or equal to two hops away can affect a PE in the 10 to 20 time window because such an event would have to have a time-stamp earlier than the start of the window.

An important question is which method will be used for determining the size of the time window. If the window is too small, there will be too few events available for concurrent execution. On the other hand, if the window is too large, the simulation mechanism behaves in a manner similar to that when no time window is used. This is equivalent to implicitly assuming an infinitely large time window implying that the overhead to manage the window mechanism is not justified. Setting the window to an appropriate size requires application-specific information that must be obtained either from the programmer or from monitoring the simulation at run-time.

Another conservative windowing protocol that Nicol [Nic91] proposes is in many aspects similar to the Bounded Lag algorithm proposed by Lubachevsky. In this protocol, each PE may advance its time up to a global ceiling. Calculation of the ceiling differs from the one proposed by Lubachevsky. Sokol, Bristo and Wieland [SBW88] propose the Moving Time Window (MTW) approach, in which a dynamically adjusted global time window is assigned to the nodes. Events within the time window are assumed to be simultaneously processable. An anomaly occurs when a node schedules an event earlier than the latest executed event at the recipient node. However, as the precedence constraints of the traditional sequential simulations are relaxed, the results of the MTW scheme is not necessarily similar to the results of the sequential one.

In the Conservative Time Windows (CTW) approach [AR94], the system to be simulated is partitioned into n disjoint subsystems, each of which is represented by an object. The scheme identifies a time window for each object such that events

belonging to different windows are independent and can be processed concurrently. The size of each window is calculated in each iteration of the algorithm using features of the system being simulated. Thus, different windows have different sizes if the nodes are advancing heterogeneously. As compared to the Bounded Lag approach and Nicol's approach, the upper end of the windows in the CTW approach is not bounded by a ceiling.

The Conservative Superstep Protocol [CLT97] proposed a technique wherein the simulation proceeded in a series of supersteps, where each superstep is followed by a barrier synchronization. Each superstep consists of two phases - the computation phase and the communication phase. The communication phase involves the communication of event messages among the participating PEs and the computation phase involves the calculation of the safe-time by the PEs. In this technique, the concept of supersteps is similar to that of a 'window' in which events are guaranteed to be causally safe. Several refinements to the Conservative Superstep Protocol were proposed in [GLC+01] based on the way the safe-time of each PE is calculated.

A good reading on various conservative synchronization algorithms can be found in [Fuj00, Nic90].

2.3 Some Definitions

In this Section we introduce certain terms that will be used frequently throughout this work. A list of the symbols used is shown in Table 2.1.

Symbol	Explanation
$\kappa[x, y]$	*Event Coverage* for the time interval $[x, y]$
$\delta[x, y]$	*Event Density* for the time interval $[x, y]$
L_i	Lookahead value of a participating PE_i
λ	Mean of a random Poisson variable
χ^2 Test	A test that examines the closeness between a chosen model and experimental data

Table 2.1: Symbols used for Explaining Super-step model and Optimization Techniques

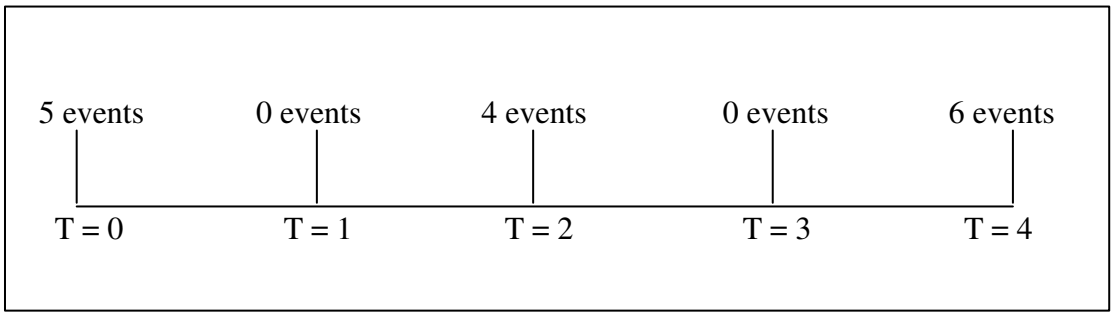

Figure 2.2: Hypothetical Sequence of Events at an Entity

2.3.1 Event Coverage

Event Coverage is the ratio of the number of time units at which the entity under observation has an event destined for it to the total number of time units in the observation period. We denote *Event Coverage* by $\kappa[x, y]$ where x and y are the lower and upper end points of the observed time interval. If the entity has events to be executed in each time unit in the observed time interval, then its *Event Coverage* is 1 for that time interval. Alternately, if the entity has no events to be executed in the observed time interval, then its *Event Coverage* is 0 for that time interval. It is quite obvious that *Event Coverage* can also be viewed as the probability that a time-step has events to be executed in a time interval and therefore ranges between 0 and 1 under all circumstances. Hence, $0 \leq \kappa[x, y] \leq 1$.

Event Coverage of an entity, is an attribute that might change based on the time interval being observed. An entity with $\kappa[x_1, y_1] = 0.2$ for the interval $[x_1, y_1]$ may have $\kappa[x_2, y_2] = 0.8$ for the interval $[x_2, y_2]$.

Consider the hypothetical distribution of events at an entity in 5 consecutive units of simulation time as shown in Figure 2.2. The calculation of the $\kappa[0, 4]$ is shown below:

$$\begin{aligned} \kappa[0,4] &= Average\ event\ coverage\ for\ the\ interval\ [0,4] \\ &= \frac{Time\ units\ in\ [0,4]\ with\ events}{Total\ Time\ Units\ in\ [0,4]} \\ &= 0.6\ (since\ 3\ time\ units\ out\ of\ 5\ contains\ events) \end{aligned}$$

2.3.2 Event Density

Event density of a PE is the average number of events received by a PE per unit simulation time during an observed simulation time interval. We represent *Event Density* using δ throughout our work. Again referring to Figure 2.2 we calculate the *Event Density* (δ) for the interval $[0, 4]$ as follows:

$$\begin{aligned} \delta[0,4] &= Average\ event\ density\ for\ interval\ [0,4] \\ &= \frac{Total\ Events\ in\ [0,4]}{Number\ of\ time\ units} \\ &= \frac{15}{5} = 3\ events\ per\ time\ unit \end{aligned}$$

2.3.3 Lookahead

Distributed logical time simulations, in general, utilize a non-negative quantity [Fuj97] called lookahead [Fuj89, Fuj90]. If a PE at logical time T has a lookahead of L, any event scheduled by the PE must have a time-stamp of at least $T + L$. Lookahead is used to define a lower bound on the time-stamp of messages produced by a PE later during the execution ($T + L$ in this example) since the current logical time of a PE can never decrease. By computing a minimum of this lower bound across all PEs that can send messages to another PE S, a lower bound on the time-stamp of messages that will be delivered to S in the future can be computed. This lower bound is important because it means S can process messages with time-stamp smaller than this lower bound with an assurance of not receiving a smaller time-stamped message in future, which would violate the constraint that all events be processed in time-stamp order.

Lookahead values of a PE can change dynamically during the execution. If the value of lookahead for a PE changes from L_{prev} to L_{new}, then any new event generated by this PE must now have a time-stamp of at least $T + L_{new}$. If $L_{prev} < L_{new}$, then the new event that can be scheduled is farther away into the future and thus none of the other PEs need to consider the events already declared as safe. However, if

$L_{prev} > L_{new}$, then the PE can schedule events that have time-stamps less than the safe-time informed to other PEs. This means that lookahead cannot instantaneously be reduced but may be increased.

2.3.4 Poisson Distribution

The Poisson Distribution is a discrete distribution which takes discrete values X, such that $X \geq 0$. It is often used as a model for the number of events (such as the number of telephone calls at a business or the number of accidents at an intersection) in a specific time period. The Poisson distribution is determined by one parameter λ, which is the mean of the distribution. The distribution function (probability mass function) for the Poisson distribution is:

$$p(X, \lambda) = \frac{e^{-\lambda}\lambda^X}{X!};\ for\ X = 0, 1, 2... \tag{2.1}$$

2.3.5 Goodness-of-fit

A maximum-likelihood fit to data provides an approximation for experimental data. Inserting the estimates into the likelihood function yields a distribution that models the data. Goodness-of-fit is a quantitative measure which indicates the closeness of the model with respect to the experimental data.

If the data are represented by (integer) numbers of events in discrete classes, Poisson statistics are commonly used to model the event occurrence. Pearson's chi-square (χ^2) Test and the likelihood-ratio test are two well established methods of dealing with this case. These tests work best when the expected number of events in a class (λ) is large, where λ is the mean of the Poisson Distribution [Man97, FK87].

2.3.6 Chi-Square Test

The sum of squares of several unit normal variates has a distribution known as Chi-Square [Jai91]. A continuous random variable x, is said to follow a Chi-Square distribution if its Probability Density Function (pdf) is of the form:

$$f(x) = \frac{1}{\Gamma(\frac{r}{2})2^{\frac{r}{2}}} x^{\frac{r}{2}-1} e^{\frac{-x}{2}}; \quad where\ 0 < x < \infty \tag{2.2}$$

where Γ, the Gamma function is given by:

$$\Gamma(\Theta) = \int_0^\infty x^{\Theta-1} e^{-x} dx; \quad where\ \Theta > 0 \tag{2.3}$$

The χ^2 test formalizes the intuitive idea of comparing the histogram of the data to the shape of the candidate density or mass function [Man97, FK87]. The test procedure begins by arranging the n observations into a set of k class intervals or cells. The test statistic is given by:

$$\chi_0^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \tag{2.4}$$

where O_i is the observed frequency in the i^{th} class interval and E_i is the expected frequency in that class interval. The expected frequency for each class interval is computed as $E_i = n * p_i$, where p_i is the theoretical, hypothesized probability associated with the i^{th} class interval.

The number of independent terms in an expression is known as its degrees of freedom. It can be shown that χ_0^2 approximately follows the chi-square distribution with $k - s - 1$ degrees of freedom, where s represents the number of parameters of the hypothesized distribution estimated by the sample statistics. The hypotheses are:

- H_0 : The random variable X conforms to the distributional assumption with the parameters given by the parameter estimates
- H_1 : The random variable x does not conform

The null hypothesis, H_0, is rejected when the calculated value of χ_0^2 exceeds the critical value derived from the chi-square statistical tables or calculated directly from the pdf of the chi-square distribution.

Chapter 3

Optimization Using Piggy-backing and Using Lookaheads

In time-stepped simulations which rely on a central Host to control the synchronization, the Host typically sends *Clock Advance* signals to all the PEs at each time-step. Though this mechanism results in good performance when the event coverage is high it can be inefficient when events are less frequent (when compared to the size of time steps). Further, the traditional time-stepped technique ignores all lookahead information (if available) as global synchronization is performed at every time-step. In this chapter, we investigate the possibility of informing the Host about future events and of utilizing lookahead information before advancing the logical time of participating PEs, in order to reduce overhead messages and time.

3.1 The Piggy-backing technique

In conventional time-stepped simulations, the Host has no information about the future events at the local Pending Event Lists (PELs) of the PEs. Hence, there is no scope for implementing an optimization to reduce synchronization costs. We aim to optimize the synchronization by informing the host about future events. This information is piggy-backed on the *Ready* messages sent by the participating PEs to the host [TTS03].

3.1.1 Motivation

In the physical world, events that alter the states of the physical entities do not occur at each and every instant of time. Typically, there are time periods when the state of a physical system remains unchanged. We can call this the inter-event time. Some systems such as traffic light systems, have a high average inter-event time as compared to others such as telephone switching boxes which have a low average inter-event time.

Defense simulations normally have some intervals of time when inter-event time may be quite low (typically during pre-war, post-war and at-war time scenarios) and high inter-event time at others (during peace, patrol and movement time scenarios). Time-stepped simulations appeal to the defense community mainly due to two reasons: they can be easily implemented to obey real-time constraints and they provide global state information at regular and frequent points in simulation time [TS02].

However pure time-stepped simulations often rely on expensive hardware for time-efficient progress due to real-time requirements and the sensitivity of such simulations to the synchronization frequency and the frequency of simulation events.

3.1.1.1 Constraints Imposed by Real-Time Simulations

In time-stepped simulations, the size of each time-step, Δ, is the most important simulation parameter. The step-size Δ is limited due to hardware considerations and also due to the architecture of the simulation. For example, in the traditional time-stepped technique a time-advance cannot be performed until all processes have signalled their readiness for the advance. If some entities are being simulated by processes running in geographically distant locations then the inter-process communication might be performed on a LAN or a WAN. This inter-process communication speed heavily limits the step-size in a time-stepped simulation, because the step-size needs to be at least as large as the maximum round-trip travel time of simulation events.

3.1.1.2 Sensitivity to Event Coverage

Time-stepped simulations tick through each participating entity at every time instant regardless of whether there are events at that particular entity or not. From this it is quite obvious that this simulation technique is well suited when entities have at-least one event on average to be executed at every instant of time. This requirement can be easily met by scaling the simulation time such that each unit of simulation time corresponds to a large interval of real time. That will increase the probability that at-least one event is present in each time-step. However, increasing the time-step interval indiscriminately will not meet another important requirement of mission-critical simulations like the ones used for simulating traffic lights and wars. By increasing the step-size we lose the ability to implement fine-granularity. Hence the optimal solution is one that serves two conflicting requirements:

- An unit of simulation time should be small enough to meet the real-time requirements
- Each unit of simulation time should be large enough to have at least one event to be executed by the entity being ticked.

3.1.1.3 Sensitivity to Synchronization Cost

Simulations using time-stepping need to synchronize at every unit of simulation time unless some form of lookahead is used. Before an advancement of time can be made at any of the entities it should be confirmed that all other entities are ready for the advancement.

Synchronization at each time-step is an over-kill if fine granularity needs to be maintained. Obviously, a decrease in the number of synchronization events is desirable but not at the cost of coarse granularity. The following section proposes the piggy-backing technique which serves this requirement.

3.1.2 Discussion of the Technique

The traditional time-stepped technique relies on the Pending Event List (PEL) to store the events generated (and to be executed in the future) at each PE. At each time-step the processed events are cleared from the PEL. To implement the piggy-backing technique, we introduce a new Future Time List (FTL) at the Host, in addition to the PEL that is maintained at the PEs. Figure 3.1 outlines the steps taking place at the PEs and the Host when optimization using piggy-backing is implemented.

At the Host:

```
while (simulation progress is not stopped by user)
{
   - scan through the FTL and calculate the minimum time-stamp 'm' in the FTL
   - send 'Time Advance to m' to the PEs corresponding to 'm' in the FTL
   - clear the PE list corresponding to 'm' in the FTL
   - do
          wait until a Ready signal is received from a participating PE
          extract piggy-backed information from the Ready event and update FTL
     while (responses from all participating PEs have not yet been received)
}
```

At the PEs:

```
while (simulation progress is not stopped by user)
{
   - wait until Time Advance is received from Host
   - process the events corresponding to the time-stamp declared safe by the Host
   - clear the processed events from PEL
   - new events generated are added to the event queue (PEL)
   - send Ready signal to Host piggy-backed with information on new events generated
}
```

Figure 3.1: Simulation Algorithm for Optimization Using Piggy-backing

A simulation execution using the piggy-backing technique involving two PEs and a central Host is now explained in detail. Let E_1, E_3 and E_4 be the local entities residing at PE_1 and E_2 and E_5 be the ones residing at PE_2 respectively. An event is represented by the notation $E_x \# E_y$, which implies that the event is generated by E_x and is destined for E_y.

To initiate the simulation the Host adds a *Start* event destined to all the participating PEs in its FTL and then dispatches it to all the PEs as shown in Figure 3.2.

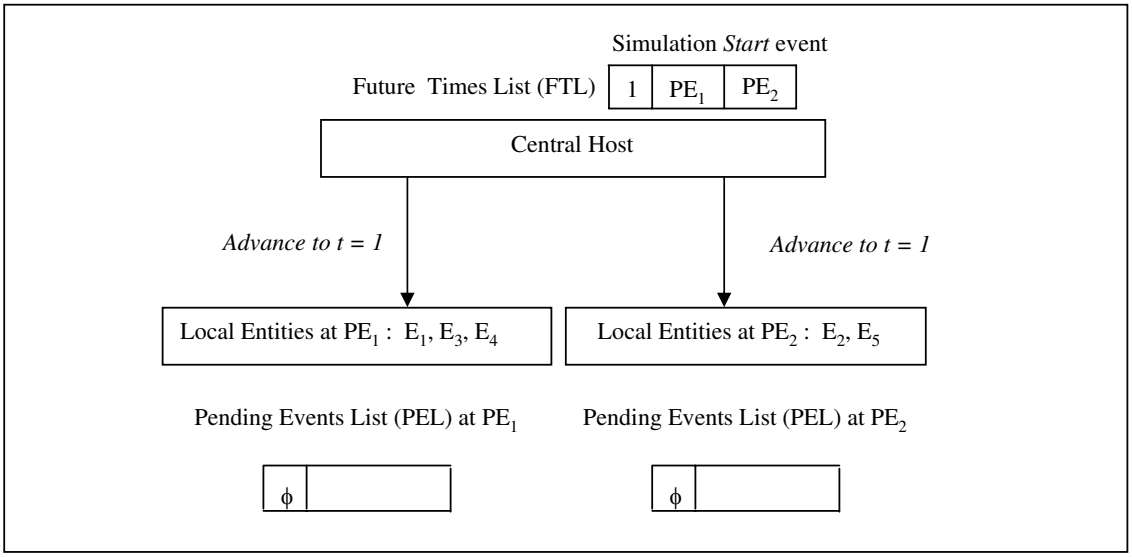

Figure 3.2: Initiation of the Simulation by the Host

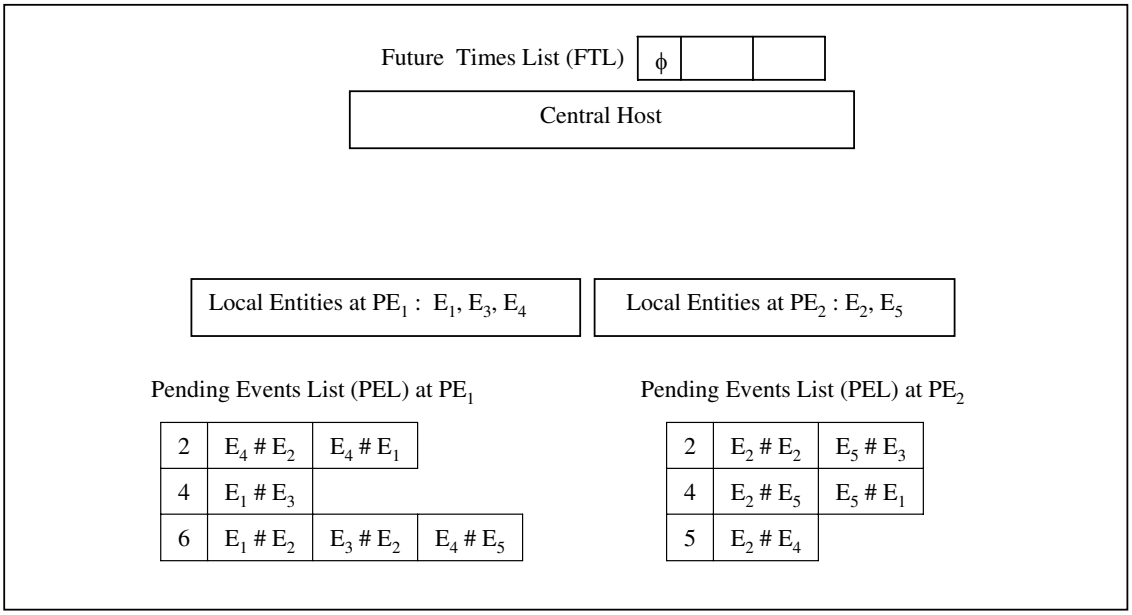

Figure 3.3: PELs with Newly Generated Events

When PEs receive the *Start* message from the host they scan through the PEL and execute the residing local entities. This might result in new events being generated which are meant to be processed during future times. The generated new events are added to the PEL. Figure 3.3 shows the PELs of PE_1 and PE_2 after the insertion

of newly generated events.

Once all the local entities have been executed the PE constructs the *Ready* message, with information about the new events that have been spawned, to be sent to the Host. This information is the time-stamps that destination PEs need to be informed about in order to receive *Time Advance* signals from the Host corresponding to those time-stamps. After constructing the piggy-backed message the PE sends out all the Events in its PEL which are not destined to itself, to the actual destination PEs . Once these non-self-destined events have been sent out, the PE clears these out from the PEL. It is crucial that this is done before the *Ready* message is sent out to the Host because sending of the *Ready* message implies that the PE is completely ready to receive the next *Time Advance* signal from the Host.

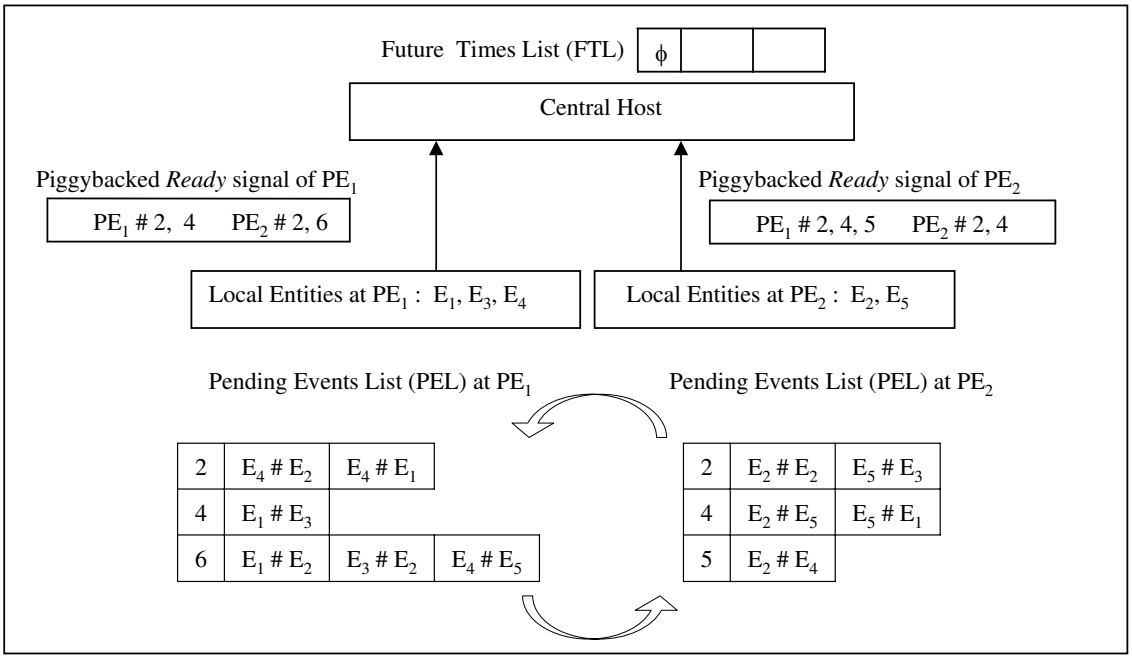

Figure 3.4: Exchange of Events and Sending of Piggy-backed *Ready* Signals from PEs

The exchange of non-self-destined events as well as the sending of the piggy-backed *Ready* message is shown in Figure 3.4. A problem could arise if *Ready* messages are sent out by PEs before the exchange of non-self-destined events as the next *Time Advance* signal from the Host could reach the PE even before it has received all the events destined to it from other PEs. Once a PE has finished with sending out

the non-self-destined events, it sends out the piggy-backed *Ready* message. It is to be noted that at the instant before sending out the *Ready* message, a PE has only those events that are destined to itself.

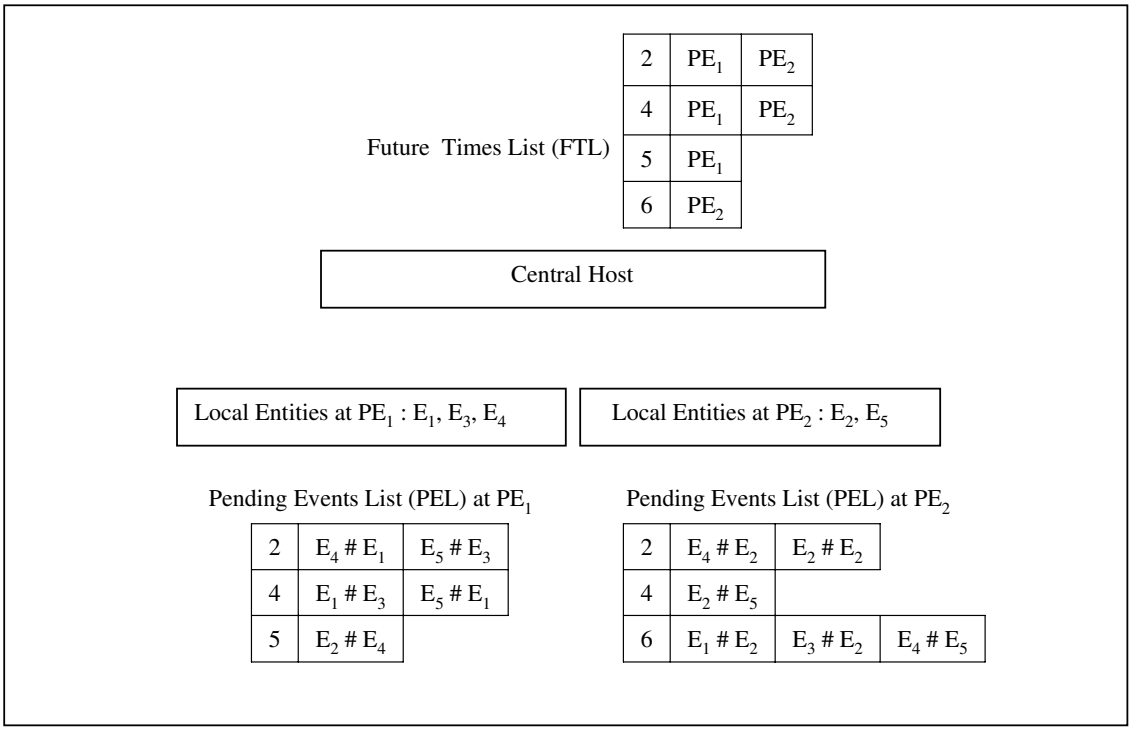

Figure 3.5: Updating of FTL by Host

The Host then extracts the piggy-backed information on receiving the *Ready* message and updates its FTL. Figure 3.5 shows the FTL at the Host after it has been updated. The Host then sends *Advance to* t where t is the minimum time-stamp in its FTL, to those PEs corresponding to time t in the FTL. In this case, the minimum time-stamp in the FTL is 2 and the PEs corresponding to 2 are PE_1 and PE_2. Hence, the Host sends an *Advance to t=2* message to PE_1 and PE_2, as shown in Figure 3.6. Once the PEs have been informed about the advance, the list corresponding to $t = 2$ in the FTL is emptied.

After receiving the *Advance* message from the Host, the PEs scan through the PEL, execute the residing local entities and insert the newly spawned events in its PEL as shown in Figure 3.7.

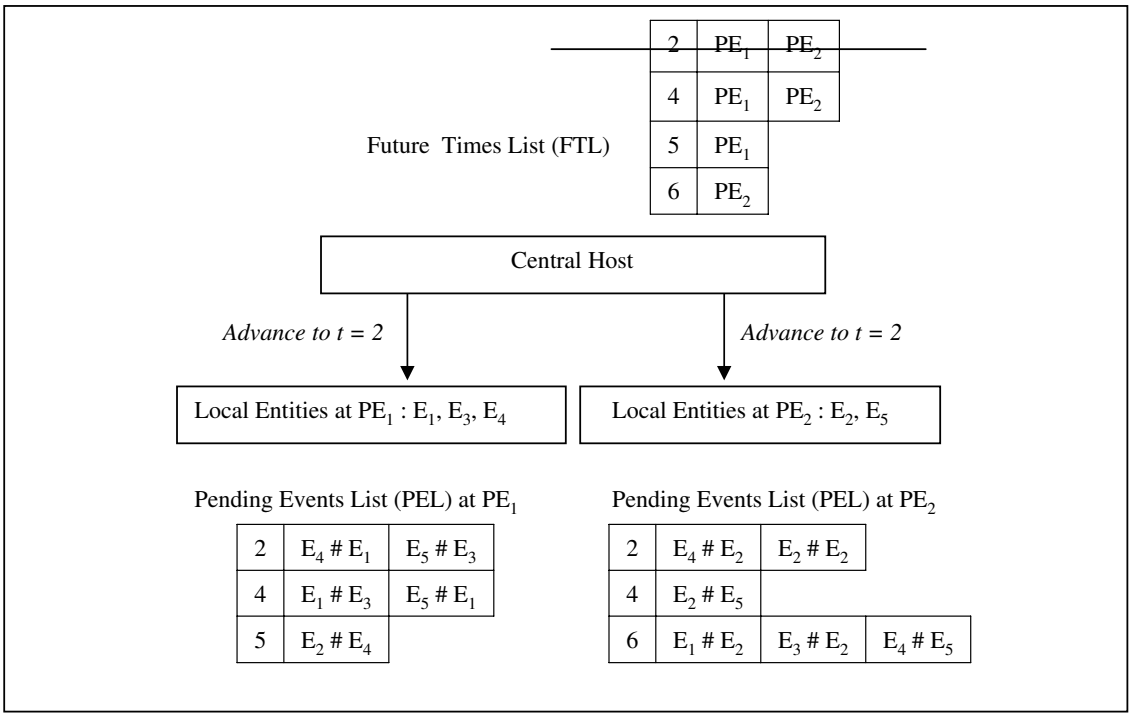

Figure 3.6: *Advance* Signal Being Sent to PEs

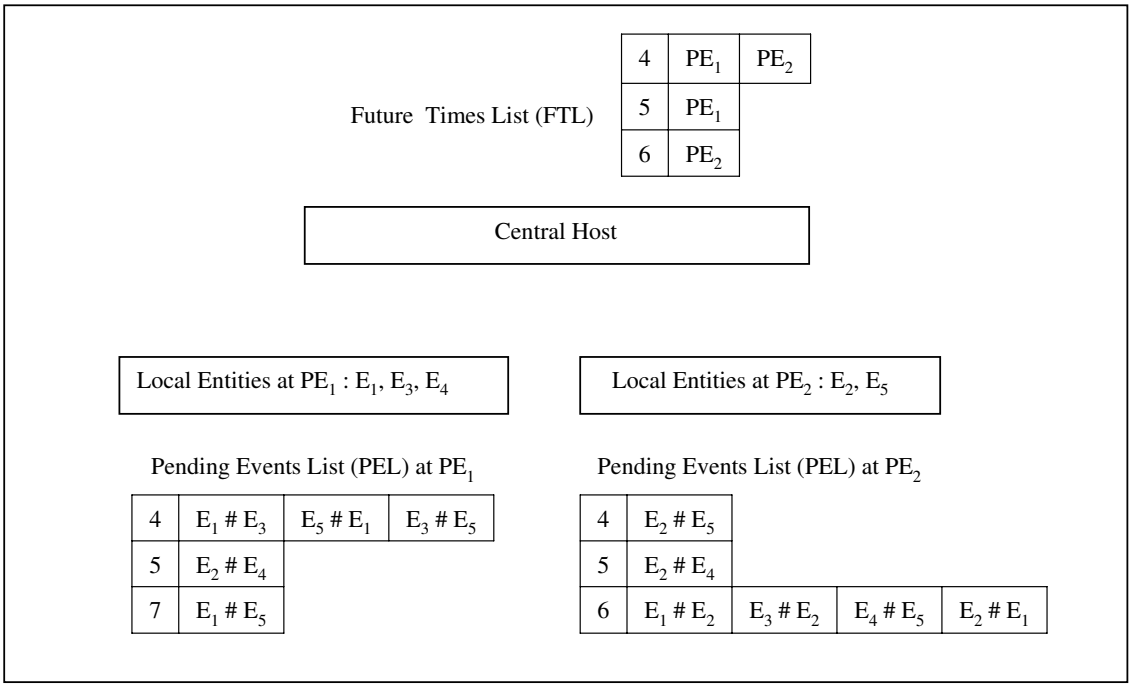

Figure 3.7: Flushing of PELs and Appending of New Event Information at PEs

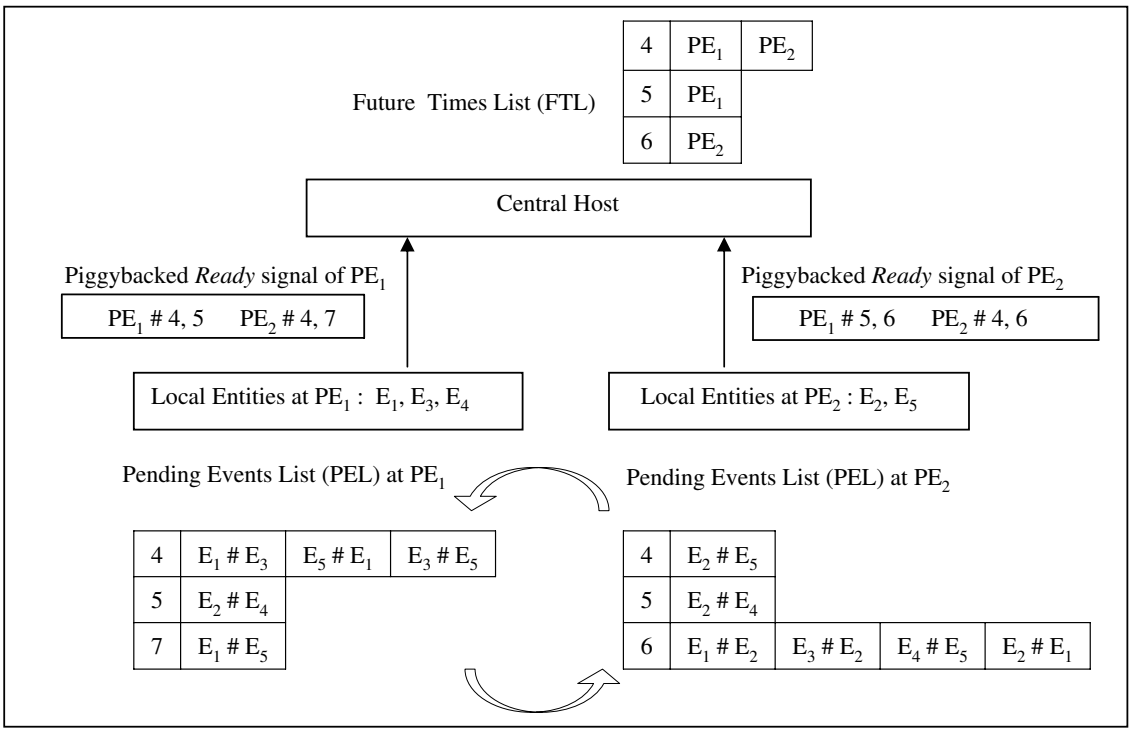

Figure 3.8: Exchange of Events and Sending of Piggy-backed *Ready* Signals from PEs

The piggy-backed *Ready* signal is then constructed before sending out all the non-self-destined events to other PEs. Finally, the *Ready* signal is sent to the Host again which is depicted in Figure 3.8.

It is to be noted here that the Host only waits for *Ready* messages from those PEs that had been sent *Advance* messages during the previous time-step. This process of the Host sending *Advance* signals and then waiting for *Ready* signals from the PEs repeats until the simulation is terminated.

When the Host receives information that it already possesses in its FTL it ignores the information. For instance, in Figure 3.8 the Host is already aware of the fact that PE_1 and PE_2 need to be sent an *Time Advance* signal when $t = 4$. Hence, the information that PE_1 and PE_2 need to be informed about time-stamp 4 in the new piggy-backed *Ready* signal being received by the Host is ignored.

Figure 3.9 shows the FTL at the Host after the information piggy-backed on the previous *Ready* signal has been collated by the Host. It can be seen that all PELs

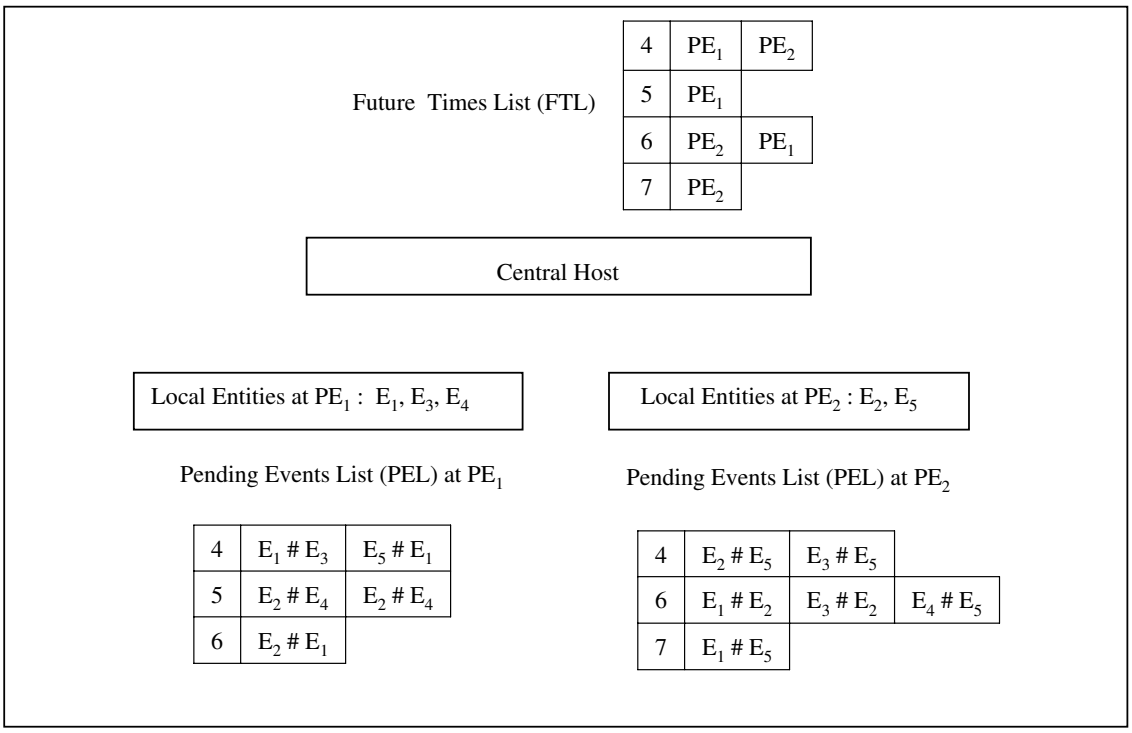

Figure 3.9: Updating of FTL by Host

only contain events destined for their respective PEs because all non-self-destined events were sent out to other PEs before sending out the *Ready* signal in Figure 3.8.

3.2 Simulation Using Lookaheads

Most real-world systems involve interactions between sub-systems. These sub-systems usually possess lookahead information that can be used to guarantee event safety for execution. If a PE is not allowed to schedule an event bearing a time-stamp within the lookahead interval, we can allow concurrent delivery and processing of messages within this safe interval. Since the PE is able to predict attribute updates and interactions at least L time units ahead of time, L is referred to as the lookahead.

3.2.1 Motivation

Typically, in a simulation which consists of a number of participating PEs, each PE cycles through the following steps:

- Determine the events that are safe to process

- Process the events identified as safe and exchange messages
- Participate in a global (or barrier) synchronization

It is to be noted that events generated in one cycle are not eligible for processing until the next cycle. The above steps give rise to two main issues *viz.* the barrier mechanism to be used and the determination of safe events.

In this approach we identify a set of future events, possibly with different time-stamps as safe rather than identifying just those events with the next minimum time-stamp. This is made possible due to the available lookahead information. PEs do not need to be informed of each time-step by the central Host and this leads to savings in synchronization messages. The downside is the barrier synchronization that is required before calculating the next set of safe events which tends to be expensive if performed frequently. Additionally, simulation using lookaheads does not conform well to strict real-time constraints.

3.2.2 Barrier Synchronization

When a process invokes the barrier primitive, it will block until all other processors have also invoked the barrier primitive. When the last process invokes the barrier, all processes can proceed further in simulation time. Barrier Synchronization is required because the calculation of the next set of safe events assumes that there are no events in transit and that all events have reached their destination.

This is possible only when global (or barrier) synchronization are performed to identify the set of safe events. There are many methods in which barrier synchronization can be achieved, such as centralized, tree based, and butterfly based methods.

3.2.2.1 Centralized Barriers

In this simple approach one of the nodes in the simulation is designated as the controller. A synchronization message is sent by all the participating PEs in the

simulation to this global controller when they enter the barrier. Once a PE enters a barrier it ceases to generate new events and the global controller releases all the PEs once it has ensured that there are no transient messages in the network and that all the participating PEs have reached the barrier. This simple technique is depicted in Figure 3.10.

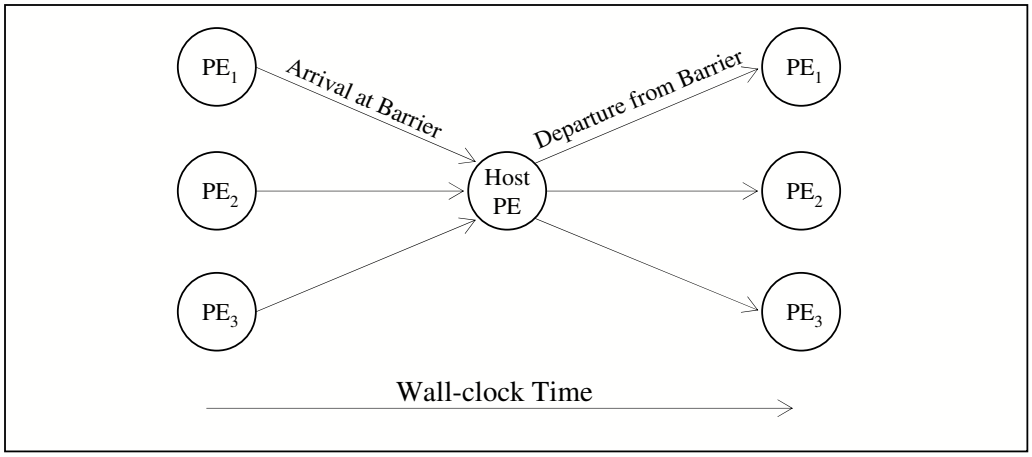

Figure 3.10: Centralized Barrier Synchronization

Like most centralized techniques the centralized barrier technique suffers from lack of scalability as the controller needs to perform $N - 1$ send and receive operations on each barrier where N is the number of participating nodes in the simulation.

3.2.2.2 Tree Barriers

The scalability problem discussed in the centralized barrier technique can be mitigated to some extent using tree barriers. In this technique, the processors participating in the simulation are organized as a balanced tree with each node of the tree representing a different processor. A child sends a synchronization message to its parent on encountering the barrier. A parent waits till it receives such a message from each of its children before propagating this message to its parent. The barrier is achieved when this message propagation reaches the root. Subsequently, the root propagates the release message downwards through the tree exactly opposite to the flow of messages used in achieving the barrier as shown in Figure 3.11 .

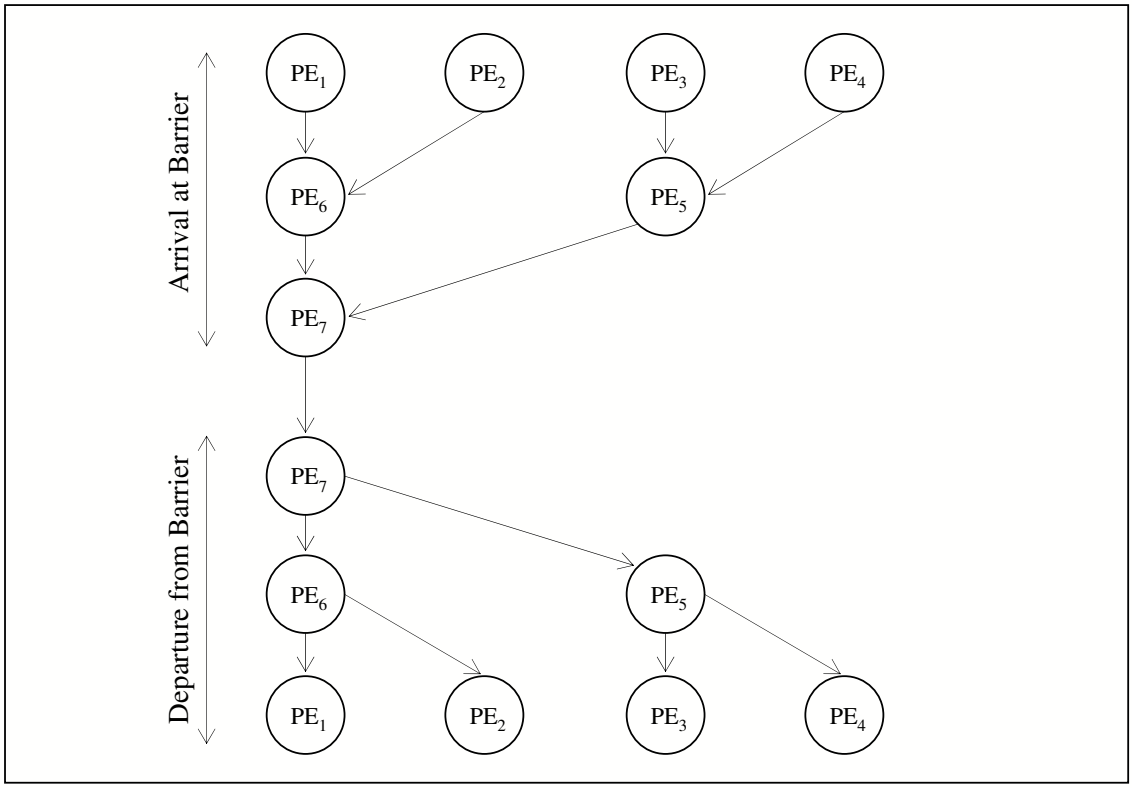

Figure 3.11: PEs Synchronizing using a Tree Barrier

The analysis of this technique reveals that it requires approximately $2 * \log_k N$ time-units to complete the barrier and requires $2 * (N - 1)$ messages, where k is the degree of the tree used and N is the number of participating nodes in the simulation.

3.2.2.3 Butterfly Barriers

In this complicated mechanism each processor executes a sequence of $\log N$ pairwise barriers with a different processor at each step. A pairwise barrier between two processors is accomplished by having one of them send a message to the other when it has encountered the barrier and then wait for a similar message from the other. To begin with, processors whose binary address differ only in the least significant bit perform a pairwise barrier. Generally, in the k^{th} step a processor synchronizes with those processors whose binary addresses differ in the k^{th} bit. Each processor is said to have completed the barrier when it has completed the $\log N$ pairwise barriers. The pairwise barriers that occur when a 8 PEs synchronize using the Butterfly barrier is shown in Figure 3.12.

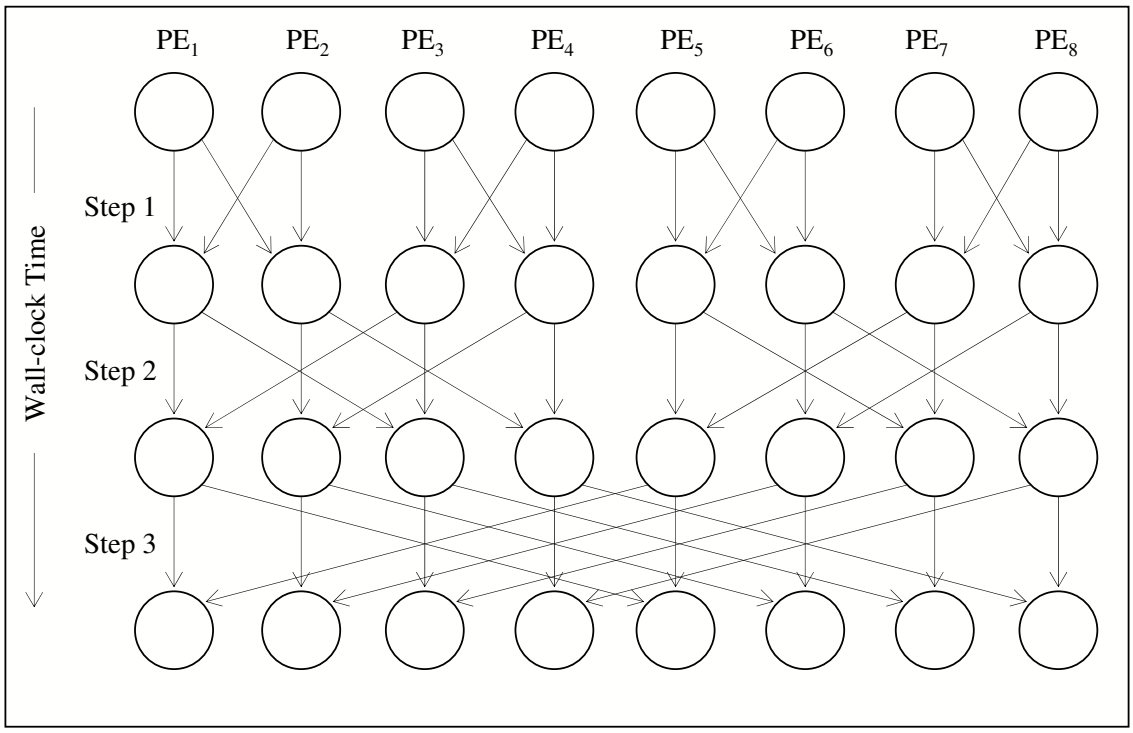

Figure 3.12: Butterfly Barrier Synchronization

It can be shown that this technique results in a successful barrier but the proof is beyond the scope of this thesis.

We use the centralized barrier in our experiments due to its ease of implementation. Further, our experiments use up to a maximum of 8 participating PEs and hence the centralized barrier technique will not pose us any significant scalability issues.

3.2.3 Discussion of the Technique

The Host maintains information about the current values of the lookahead for each participating PE. Additionally, the Host is also aware of the simulation time to which each PE has progressed. Figure 3.13 outlines the steps taking place at each PE when Optimization using Lookaheads is used with centralized barriers. At each barrier, the Host informs the next safe-time to each PE by calculating the minimum value of $(T_i + L_i)$ where T_i is the simulation time at PE_i and L_i is its lookahead. The time interval between the current time and the safe-time informed by the Host forms a 'window' in which events are guaranteed to be causally safe.

At the Host:

while (simulation progress is not stopped by user)
{
- for each PE participating in the simulation
  calculate $ST_i = (T_i+L_i)$
- from values of ST_i above
  calculate ST = minimum (ST_i for all i)
- send '*Time Advance to ST*' to the PE
- do
  wait until a *Ready* signal is received from a participating PE
  while (responses from all participating PEs have not yet been received)

}

At the PEs:

while (simulation progress is not stopped by user)
{
- wait until *Time Advance* is received from Host
- process the events with time-stamps <= time-stamp declared safe by the Host
- flush the processed events from PEL
- send Ready signal to Host

}

Figure 3.13: The Lookahead Based Technique

Many improvements have been proposed to this simple synchronous technique as discussed in Section 2.2. However, our main purpose of introducing the optimization using lookaheads is to develop the super-stepping technique which is presented in detail in the next chapter on super-steps. Hence, we use the simple synchronous technique in our experimental evaluation due to its ease of analysis and implementation. The progress of a simulation that uses barrier synchronization is shown in Figure 3.14.

The lookahead based technique requires a slight relaxation of real-time constraints. Since we are limiting ourselves to time-stepped simulations we know that the simulation engine assumes all events generated within a time-step bear the same time-stamp. This further means that the size of the time-step dictates the frequency at which the user gets to visualize the current state of the simulation.

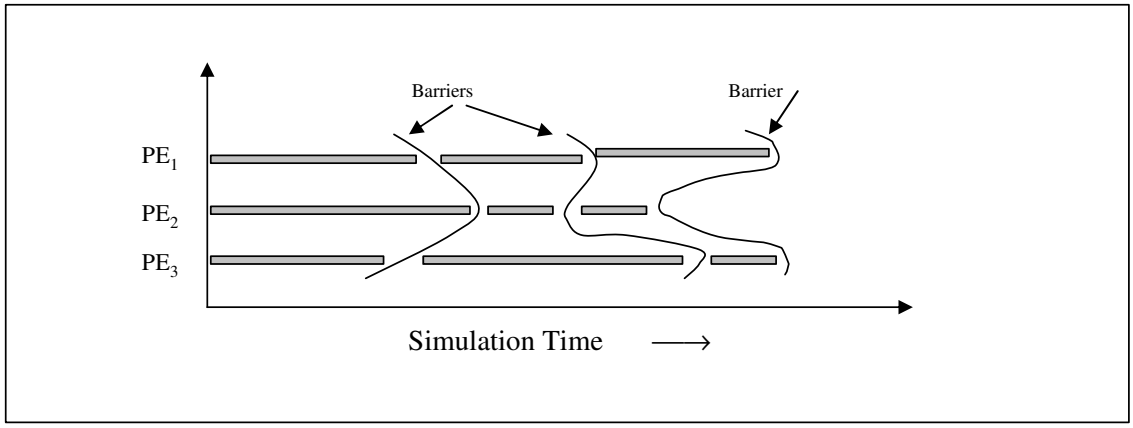

Figure 3.14: Simulation Progress using Barrier Synchronization.

Though a small time-step ensures a very fine-resolution simulation and provides the user with an almost continuous feel, it causes a very high overhead resulting in great difficulties in meeting real-time deadlines. Hence simulations normally employ a balanced time-step size which gives sufficient reality to the user and generates a manageable number of overhead messages.

In the piggy-backing based optimized technique, the user receives an up-to-date picture of the simulation state at all points of time because if any event with a particular time-stamp t exists, it will not only be processed within $t + \Delta$ but the display of the simulation state will also be updated immediately for the benefit of the user. The savings in communication messages is achieved by piggy-backing the *Ready* messages with the list of future event time-stamps as described in Section 3.1.2. Though the technique obviates synchronization between all participating PEs at every time step, the synchronization between those PEs that have events to be processed is performed just as in the case of the conventional technique. Thus, the simulation state presented to the user is updated at every time step that has any events to be processed.

The lookahead based technique identifies "windows" in which all the events that fall within the window are free of any causality relationships. No synchronization of any sort is performed among participating PEs within a window. This technique is inadvisable for use in real-time interactive simulations due to the following reasons:

1. To provide a real-time feel and interaction ability to the user the window size needs to be kept extremely small. Hence, the lookahead information may not be used completely.

2. Since we can update the visualization of the simulation state only at window boundaries the user may not be able to see the simulation state after each event.

However this technique tends to be effective and efficient when used in case of simulations that do not have real-time interactive constraints. Typically this involves simulations that test repeatability and simulations that involve computer generated forces such as war game simulations.

Chapter 4

Super-Steps

Simulation is an inherently dynamic process and it can be difficult to predict the behavior of any of the simulation variables reliably and accurately. As the progress of a simulation is dynamic the simulation techniques can be chosen based on prevailing variable values in the simulation. This implies that it is helpful to switch between simulation techniques depending on parameters that govern the efficiency of the simulation. To allow for this we introduce the concept of super-steps. Super-steps are pre-determined points in logical time where all participating PEs will mandatorily participate in a synchronization exercise whether they have any events to be processed at that time or not. This chapter introduces super-steps, its advantages and a probabilistic, history-based method to estimate super-step sizes [TTS03].

4.1 Advantages of Super-Stepping

Introduction of super-steps helps us to switch between simulation techniques efficiently and also eases global state-saving and checkpointing. We discuss these advantages briefly:

4.1.1 Switching Between Simulation Techniques

Any switching of technique requires a particular switching point which is agreed upon by all entities. We can call this the 'clean-slate' time which means that this

time is such that the entity contains within itself all information about its state and can proceed smoothly should there be a change in simulation technique. The super-step boundary in our techniques provides the entities with such 'clean-slate' points.

Typically, an important variable to be considered is the event density. A technique that is well-suited for simulations with low event densities may prove to be inefficient when used in cases where the event density is very high. Another system variable which can affect the simulation technique efficiency is the entity lookaheads.

Use of lookaheads can result in potential benefits. However the benefits obtained by using lookahead based techniques are nullified in cases where the lookahead values are low compared to the overhead costs incurred by synchronization and lookahead information update. Thus, in some cases it might be beneficial to ignore availability of lookaheads especially when some of the participating entities have negligible lookahead values.

Communication cost of inter-process messages is another factor to be considered. Communication messages are typically intra-process, LAN-based and WAN-based depending on the geographical distances between participating processes in a simulation. It is quite evident that techniques that avoid inter-entity communication by adding a lot of complexity at each entity may not be very useful in the case where the entities are being simulated by the same PE.

4.1.2 Convenient State Saving and Strategy Manipulation

An important characteristic of simulations used for military and defense purposes is the capability to save global simulation (mission) states, which can be used for playback of specific range of frames. Introducing super-steps will help us to simplify the mechanisms that implement these features.

Wall-clock time forms the basis for the transactions and dealings in the day-to-day world. Changes in strategies are conveyed to real-world entities in real-time. Hence,

it follows that similar capabilities are desirable in simulations which depict these real-world missions. Super-step boundaries represent a time (in wall-clock time) where all PEs are exactly at the same point in time. Thus, the size of these super-steps are an important parameter which dictates the flexibility of the simulation.

The ideal case is when the average event density is sufficiently high and the standard deviation is low which implies that almost all the participating entities are having a satisfactory event density. However, when super-stepping is based on a centralized measure of mean and standard deviation it is easy to see that the calculated super-step size will be biased towards entities with high event density. Therefore, it is important to decentralize the next super-step estimation and at the same time maintain only one global super-step size at any given simulation time. This can be achieved in many ways *viz.* either by averaging out the super-step sizes estimated by each entity or by considering the maximum (or the minimum) super-step size or also by having all entities participate in a voting exercise to choose among some possible super-step sizes.

The basis used to estimate the size of the next super-step requires some analysis. It is a well-known fact that while modelling real-world phenomena it is not practically possible to know beforehand the actions of the entities participating and the events exchanged. If this was not true, results could be predicted well in advance and analysis of systems could be done mathematically or numerically. In our experiments, we maintain a randomized exchange of events between PEs in order to keep the results as unbiased as possible. By randomized we mean that it cannot be predicted beforehand whether an entity will respond to a particular event in the same time-step (immediately) or in the next or in the ones there-after.

Some form of estimation of future events is hence necessary. By sampling the event exchange phenomenon we select some appropriate statistical distribution and then try to see how well our observed data fits the distribution. If we find a reasonably good fit we estimate the next super-step size based on the properties of the distribution or else we use a very rudimentary regression approach.

The two possible choices that we consider are:

- Fit a distribution to the number of events received at an entity during the last few units of simulation time and let entities send their votes to the host regarding their opinion about the next super-step size.
- Fit a linear regression to the events received during the last few units of simulation time and as before let entities send their votes to the host regarding their opinion about the next super-step size.

As the events are occurring at random in our experimental model, independently of each other, the first approach puts us in a better position to estimate the size of the next super-step. Hence, a Poisson fit to the distribution of incoming events is more reliable than a simple regression fit.

4.2 Estimation of the Super-Step Size

A list of the symbols used in the mathematical estimation of the super-step sizes is shown in Table 4.1.

Symbol	Explanation
ψ, $\|\psi\|$	The history period and its length in simulation time units.
δ_2^t	Event Density of PE_2 in simulation time unit t.
l^n	Length of the n^{th} simulation time super-step in units of simulation time.
ζ_n^{sim}	Count of simulation events belonging to the n^{th} super-step.
$\zeta_n^{overhead}$	Count of overhead events belonging to the n^{th} super-step.
ζ_n^{send}	Overhead cost incurred by PEs while sending messages to the Host in the n^{th} super-step.
ζ_n^{recv}	Overhead cost incurred by PEs while receiving messages from the Host in the n^{th} super-step.
$\|_\psi$	Denotes that the preceding variable is an estimate based on the observed history. Eg: $\zeta_{n+1}^{sim}\|_\psi$ is an estimate of ζ_{n+1}^{sim} based on ψ.
$S\|_\psi$	Number of super-step Synchronization points in ψ
χ	The Super-step throttle
$B\|_\psi$	Number of Barrier points in ψ

Table 4.1: Symbols used for Mathematical Abstraction of Super-step Size Estimation

Let the simulation consist of a number i of PEs each simulating one or more Logical Processes (LPs). Throughout this work we will use the terms LPs and entities interchangeably to represent each individual 'actor' in the simulation. Let the PEs be represented by PE_1, PE_2, PE_3,..., PE_i respectively. δ_1, δ_2,...,δ_i represent the *Event Densities* (Refer 2.3.2) of PE_1, PE_2,..., PE_i respectively. Further, let the superscript denote the simulation time unit to which the event density corresponds. For instance the event density of PE_2 in simulation time unit t will be denoted as δ_2^t.

The average event density for a PE for a time interval consisting of a sequence of time units will be the average of event densities corresponding to each simulation time unit in that interval.

Therefore:

$$\delta_2^{t_1-t_2} = \frac{\sum_{t_1 \leq t \leq t_2} \delta_2^t}{|t_1 - t_2|} \tag{4.1}$$

Let l^n represent the length of the n^{th} simulation time super-step in units of simulation time. If the $(n+1)th$ super-step begins at simulation time t, we have

$$t = \sum_{i=1 \ to \ n} l^i \tag{4.2}$$

At each PE we have two types of events - simulation events and the overhead (or housekeeping) events. Overhead events are those events that do not actually contribute any tangible value to the simulation. They are the overheads that result from distributing the simulation onto many PEs. Synchronization costs, fault-tolerance costs are some examples of overheads. Simulation events are those that actually form a part of the simulation and modify the simulation state in some way or another.

An ideal simulation is one which has zero overhead events, but this is probably possible only in a monolithic single LP, single PE, sequential simulation. Any form

of parallel and distributed simulation setup will involve, at the very least, time synchronization overheads so that causality constraints are not violated.

Finally, since most of our predictions about the future will be based on the observed behavior from the past we denote the length of the history period considered (in simulation time units) as $|\psi|$ and the history period itself as ψ. Further, we will use the notation $|_\psi$ to denote that the preceding variable is an estimate based on the observed history. Hence, $\zeta^{sim}_{n+1}|_\psi$ is an estimate of ζ^{sim}_{n+1} based on the history period ψ.

We now estimate the next super-step size l^{n+1} based on the observed pattern of events so far. The analysis assumes that t number of simulation time units have been executed up to this point in time and since we are at a synchronization point these t time units comprise of say n number of simulation time super-steps each of variable size. A strong motivation to rely on the history to estimate the super-step size is the fact that all future events will be spawned, either directly or indirectly, by the events that have already been observed. The events that will be executed in the future will be spawned either by the events that we have already observed or due to them.

4.2.1 Optimization Using Piggy-backing

Section 3.1.2 discussed the piggy-backing based technique in detail. We extend the technique to accommodate super-stepping by making the Host mandatorily inform all the participating PEs about a super-step boundary regardless of whether there are events to be executed at that time-step or not. To arrive at an estimate for the next super-step size when the simulation is progressing using the piggy-backing based technique we first estimate the possible number of overhead events and simulation events in the next super-step based on observed history.

4.2.1.1 Estimation of Overhead Events

We abstract the following two overheads that will be used in our estimation:

1. Cost incurred by each PE during the history period in informing the centralized host about the time-stamps of future events generated by it, represented by $\sum_{n\in\psi} \zeta_n^{send}|_\psi$.

2. Cost incurred by the Host in informing a PE about the global current time, represented by $\sum_{n\in\psi} \zeta_n^{recv}|_\psi$.

The total overhead cost per individual PE, incurred across ψ, is the sum of $\sum_{n\in\psi} \zeta_n^{send}|_\psi$ and $\sum_{n\in\psi} \zeta_n^{recv}|_\psi$. Therefore,

$$\sum_{n\in\psi} \zeta_n^{overhead}|_\psi = \sum_{n\in\psi} \zeta_n^{send}|_\psi + \sum_{n\in\psi} \zeta_n^{recv}|_\psi \tag{4.3}$$

The *Ready* messages sent to the host from the entities are replied with one single safe time declaration event by the host. In addition, there might not be any event at an entity at some global synchronization points (boundary of a super-step). In the worst case, all boundaries of super-steps in the history period may not have an event to be executed at an LP. If the number of synchronization points in ψ is represented by $S|_\psi$, then:

$$\sum_{n\in\psi} \zeta_n^{recv}|_\psi \leq \sum_{n\in\psi} \zeta_n^{send}|_\psi + S|_\psi \tag{4.4}$$

Using Equations 4.3 and 4.4, we have:

$$\sum_{n\in\psi} \zeta_n^{overhead}|_\psi \leq (2 * \sum_{n\in\psi} \zeta_n^{send}|_\psi) + S|_\psi \tag{4.5}$$

We now endeavor to estimate these costs for the next super-step. In our experiments we fit a Poisson Distribution to the observed events in the history period ψ. In order to achieve this, we define the class frequency of a class $X - Y$ as the number of time-steps in ψ which had an event-density ranging between X and Y and then distribute the observed events in ψ into these classes. For example, if $1-3$ is a class then the class frequency will be the number of time-steps in ψ which had an event-density ranging between 1 and 3.

To test the Goodness-of-fit we use the χ^2 Test. As defined in Section 2.3.6 the χ^2 test involves measuring the difference between the squares of the observed class frequency and the theoretical class frequency and then calculating a goodness measure as to whether the observed frequencies match that of the theoretical Poisson Distribution. The χ^2 test can result in two possible outcomes:

- The test supports our hypothesis that the event counts fit a Poisson distribution.
- The test rejects our hypothesis.

A linear fit is used in the estimation of the number of future events when the χ^2 Test rejects the possibility of a Poisson Fit.

- Case 1: Successful Poisson Fit

  Since we have a successful Poisson fit we use the Poisson mean represented by λ .

$$\begin{aligned} \sum_{n\in\psi} \zeta_n^{send}|_\psi &= Number\ of\ Ready\ events\ sent\ to\ Host\ in\ \psi \\ &= P[Time-step\ has\ events] * |\psi| \\ &= (1 - P[\delta_i = 0]) * |\psi| \\ &= (1 - e^{-\lambda}) * |\psi| \end{aligned} \tag{4.6}$$

  For ease of analysis we reduce the result in Equation 4.5 to an equality by considering only the worst case behavior. Hence,

$$\sum_{n\in\psi} \zeta_n^{overhead}|_\psi = 2 * (1 - e^{-\lambda}) * |\psi| + S|_\psi \tag{4.7}$$

- Case 2: Unsuccessful Poisson Fit

For this case, we use a simple linear estimate. This rudimentary method does not provide us with any means of knowing what the probability that a particular entity has an event to be executed in any simulation time unit is. Hence we estimate the overhead directly based on the event coverage κ observed during the history period. Therefore,

$$\sum_{n \in \psi} \zeta_n^{overhead}|_\psi = 2 * \kappa * |\psi| \tag{4.8}$$

This is because every time-stamp that has an event involves a synchronization message and a *Ready* message exchanged between the entity and the host.

4.2.1.2 Estimation of Simulation Events

Again, we base our estimation on the observed events in the history period. Firstly, the total simulation event count in the history period is:

$$\sum_{n \in \psi} \zeta_n^{sim}|_\psi = \sum_{t \in \psi} \delta_i^t \tag{4.9}$$

This is the same as the product of mean observed event density during the history period and the length of the history period, that is:

$$\sum_{n \in \psi} \zeta_n^{sim}|_\psi \quad = \quad \begin{cases} \delta_i^{mean} * |\psi| & if\ Linear\ fit\ is\ used \\ \lambda * |\psi| & if\ Poisson\ fit\ is\ used \end{cases} \tag{4.10}$$

where δ_i^{mean} is the average event density of PE_i calculated using Equation 4.1 for the length of the history period.

4.2.1.3 Estimation of Super-Step Size

Using the above results we now calculate the length of the next super-step that will yield us the desired number of total events in a super-step. Let us assume that the simulation offers an acceptable nearness to the real world when an event count

of χ per super-step is used. χ (referred to as Super-step throttle henceforth) is then used to estimate l^{n+1} using:

$$l^{n+1} = \frac{\chi * |\psi|}{\sum_{n\in\psi} \zeta_n^{overhead}|_\psi + \sum_{n\in\psi} \zeta_n^{sim}|_\psi} \tag{4.11}$$

Equation 4.11 follows from our observation that $\sum_{n\in\psi} \zeta_n^{overhead}|_\psi + \sum_{n\in\psi} \zeta_n^{sim}|_\psi$ number of events have been observed in $|\psi|$ number of time-steps previously and we would like to see χ number of events in the next super-step whose length is denoted by l^{n+1}.

Substituting the results obtained in Equations 4.7, 4.8 and 4.10 in Equation 4.11 we obtain:

$$l^{n+1} = \begin{cases} \dfrac{\chi * |\psi|}{2 * \kappa * |\psi| + \delta_i^{mean} * |\psi|} & if\ Linear\ fit\ is\ used \\ \dfrac{\chi * |\psi|}{2 * (1 - e^{-\lambda}) * |\psi| + S|_\psi + \lambda * |\psi|} & if\ Poisson\ fit\ is\ used \end{cases}$$

We can rewrite the above equation, after simplifying, as:

$$l^{n+1} = \begin{cases} \dfrac{\chi}{2 * \kappa + \delta_i^{mean}} & if\ Linear\ fit\ is\ used \\ \dfrac{\chi}{2 * (1 - e^{-\lambda}) + \frac{S|_\psi}{|\psi|} + \lambda} & if\ Poisson\ fit\ is\ used \end{cases}$$

It is worth noting that when we consider the case where every simulation time unit has an event the estimate for l^{n+1} reduces to $\frac{\chi}{2+\delta_i^{mean}}$ and $\frac{\chi}{2+\lambda}$ respectively.

Since all the above calculations are being done at the PE level, the host will receive each PE's reply about the next preferred time super-step length. The decision is left to the host as to how to handle the disparate replies. A max, min or an average are possible alternatives. The experimental results use the average method.

4.2.2 Optimization Using Lookaheads

When super-steps are imposed on a barrier synchronization based simulation, only those PEs that have not reached the super-step boundary are involved in the barrier synchronization. PEs that have already reached the boundary are made to wait until the other PEs reach the boundary. Figure 4.1 shows a simulation involving *3* PEs progressing when using the lookahead based technique with super-stepping.

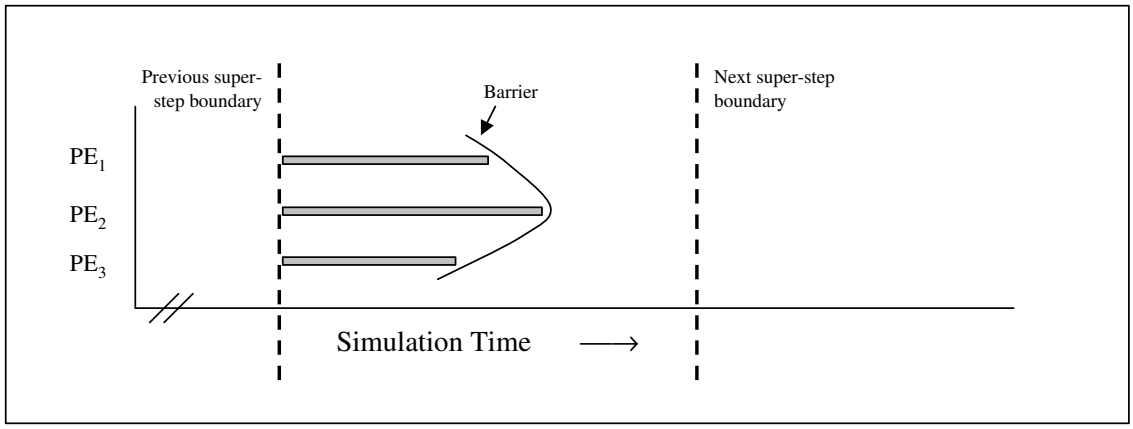

Figure 4.1: Host Informing PEs About the Next Safe-Time

All the PEs participate in the barrier, since none of them have reached the super-step boundary. Once the Host receives Ready events from all the PEs it calculates the next safe-time for all the PEs. If the time calculated as safe by the Host for a particular PE is greater than or equal to the next super-step boundary (as in the case of PE_3) then the Host informs the PE to proceed only to the super-step boundary as shown in Figure 4.2.

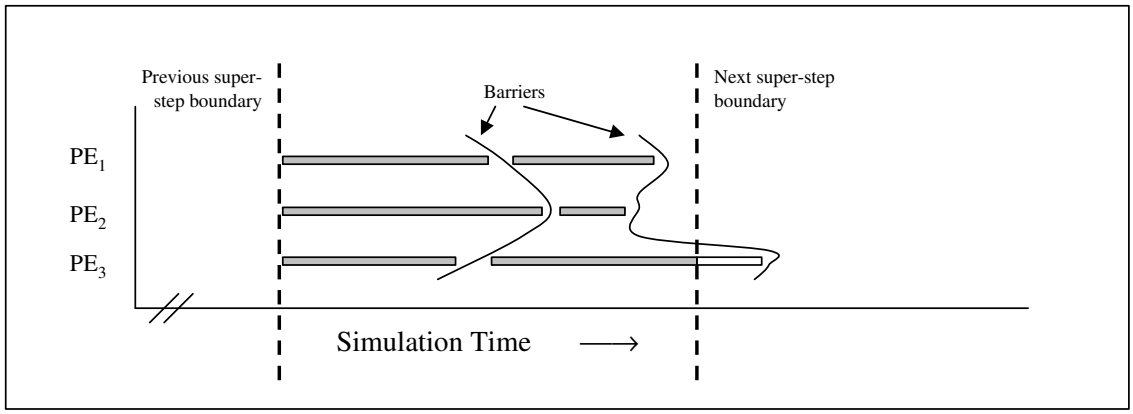

Figure 4.2: Host Ensuring that PEs Remain Within Super-step Boundary.

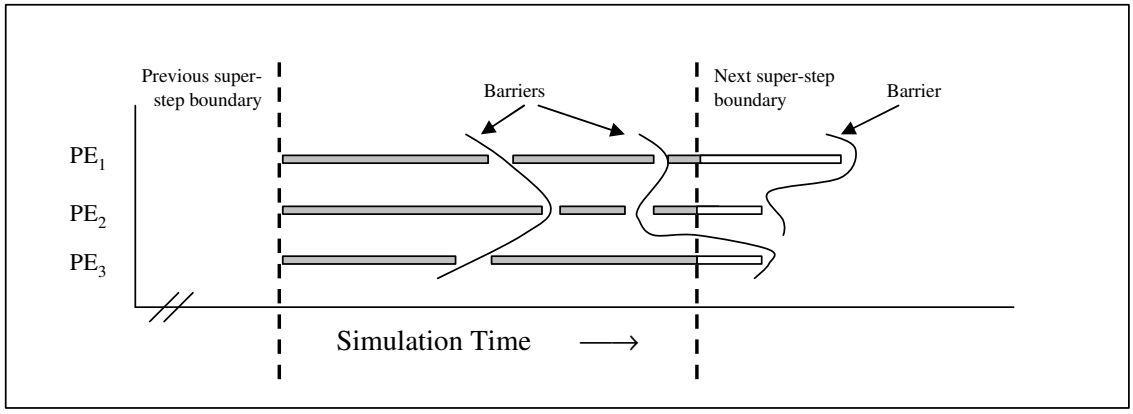

Figure 4.3: Barrier Synchronization Involving a Subset of the PEs.

Figure 4.3 shows only a subset of the PEs (PE_1 and PE_2) participating in the barrier synchronization. Since PE_3 has already reached the super-step boundary the Host dopes not involve it in the barrier synchronization. Once all PEs reach the boundary the Host permits the PEs to proceed into the next super-step. A new super-step size is calculated and then safe-times are informed to the participating PEs as shown in Figure 4.4.

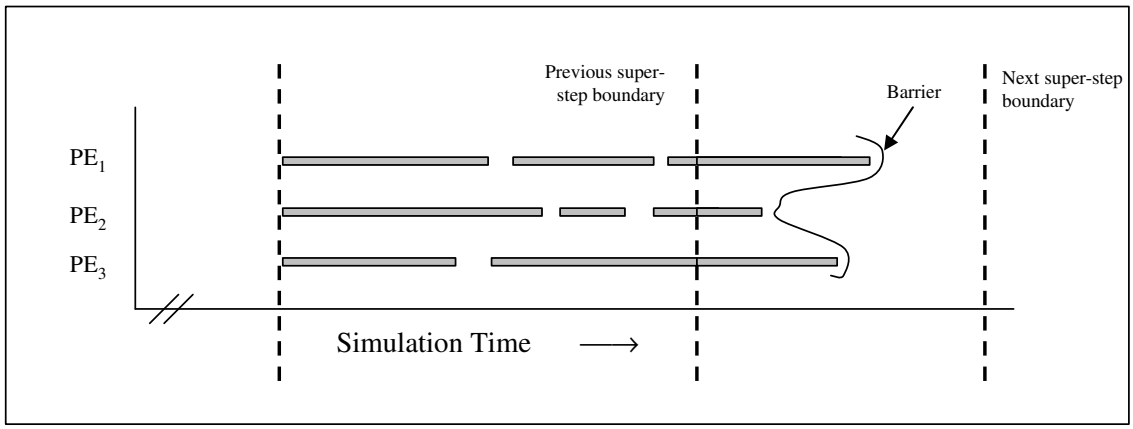

Figure 4.4: PEs Proceeding into the Next Super-step.

The next super-step size l^{n+1} is estimated based on the observed pattern of events in the recent past.

4.2.2.1 Estimation of Overhead Events

This approach involves only one overhead that is quite different in nature and cost from the previous technique, namely the cost per unit simulation time incurred in executing barriers when needed. The information about the time-stamps of future events generated by the particular entity during the previous barrier is piggy-backed on to the barrier messages and hence do not add any component to the overhead.

Estimating $\zeta_n^{overhead}$ is quite straightforward since centralized barriers are being used. The super-step boundary can also be treated as a barrier since the successful completion of the super-step boundary requires the same number of message transfers as that needed for achieving a barrier. Each PE bears an overhead of 2 messages in the process of invoking and achieving a barrier - one when informing the host that it has reached the barrier and the other when being released from the barrier by the host. Hence, the number of overhead messages exchanged during the history period ψ is:

$$\begin{aligned}\sum_{n\in\psi} \zeta_n^{overhead}|_\psi &= 2 * (Number\ of\ synchronization\ points\ in\ \psi \\ &\quad + Number\ of\ barriers\ in\ \psi) \\ &= 2 * S|_\psi + 2 * B|_\psi\end{aligned}$$

4.2.2.2 Estimation of Simulation Events

We base our estimation, as before, on the observed events in the history period ψ. Since the number of simulation events follows the same ideology as the previous technique we use the same heuristics to estimate it.

From Equation 4.10, we know that:

$$\sum_{n\in\psi} \zeta_n^{sim}|_\psi = \sum_{t\in\psi} \delta_i^t \tag{4.12}$$

4.2.2.3 Estimation of Super-Step Size

We would like to see χ number of events in the next super-step whose length is denoted by l^{n+1} and from the history we know that $\sum_{n\in\psi}\zeta_n^{overhead}|_\psi + \sum_{n\in\psi}\zeta_n^{sim}|_\psi$ number of events have been observed in $|\psi|$ number of time-steps previously. Hence, the next super-step size estimate is given by:

$$\begin{aligned} l^{n+1} &= \frac{\chi * |\psi|}{\sum_{n\in\psi}\zeta_n^{overhead}|_\psi + \sum_{n\in\psi}\zeta_n^{sim}|_\psi} \\ &= \frac{\chi * |\psi|}{2 * S|_\psi + 2 * B|_\psi + \sum_{t\in\psi}\delta_i^t} \end{aligned} \tag{4.13}$$

$\sum_{t\in\psi}\delta_i^t$ equals the product of mean observed event density during the history period, δ_i^{mean} and the length of the history period, $|\psi|$. Hence,

$$\sum_{t\in\psi}\delta_i^t = \delta_i^{mean} * |\psi| \tag{4.14}$$

Substituting Equation 4.14 in Equation 4.13, we have

$$l^{n+1} = \frac{\chi * |\psi|}{2 * S|_\psi + 2 * B|_\psi + \delta_i^{mean} * |\psi|}$$

Here again, each PE sends its next preferred super-step length to the Host and the decides the super-step length.

4.3 Super-steps Based Simulation Flow

Figure 4.5 shows a possible simulation flow when super-steps are used to facilitate switching of simulation techniques.

During the first 4 time-steps we see that the simulation progresses using traditional time-stepped technique. After that the first super-step boundary is encountered at which the next super-step size is calculated as 6 and the technique is switched to

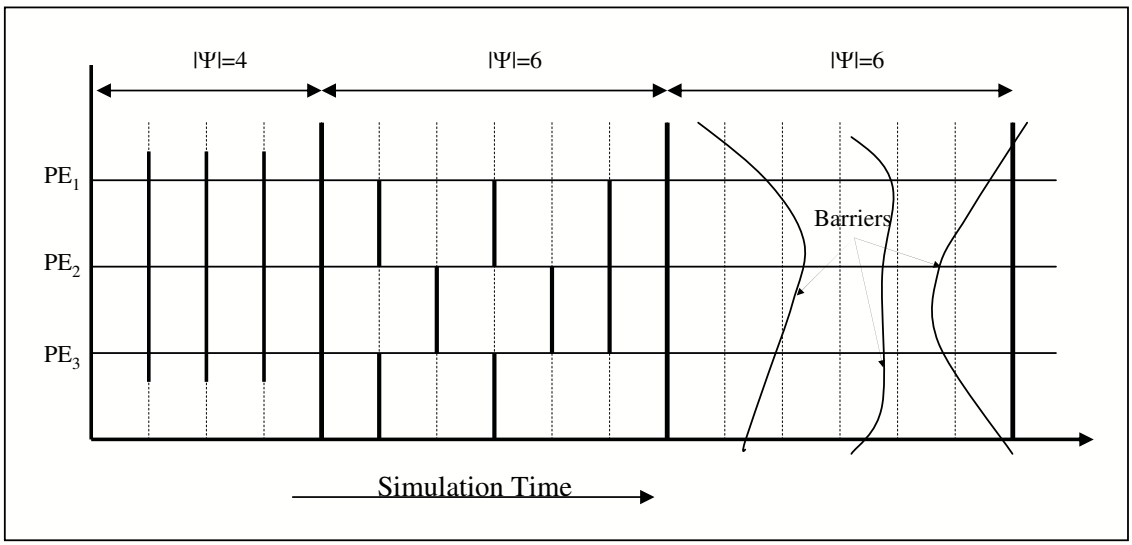

Figure 4.5: Possible Switching in Simulation Techniques

piggy-backed time-stepping. The next 6 time-steps proceed with all PEs implementing the piggy-backed technique. In the next super-step the figure shows the simulation proceeding using lookaheads.

Chapter 5

Experimental Validation

This chapter begins with detailing the various organizing principles available when transforming a conceptual model into a computer simulation. The structure of each LP participating in the simulation is then explained before presenting the results of the experiments and some observations based on the results.

5.1 Simulation Framework

When translating the conceptual model into a computer program, the modeler adopts a world-view or orientation. The world-view may be dictated by the simulation language or package chosen for the implementation or constitutes a design choice when implementing in a general-purpose language. The most common world-views for DES are: event-oriented modelling, process-oriented modelling, and activity scanning modelling [JBN96].

When adopting an event-oriented view, the modeler codes different event routines to correspond to the different event types occurring in the model. The event routines manipulate the state variables when the event occurs. Events are scheduled to occur sometime in the future either at initiation of the model or by other event routines. The set of future events, the pending event set, is maintained by the simulation system so that the next event to occur can be retrieved easily. The execution process mainly consists of extracting the next event, executing it - possibly scheduling new

events - and repeating it until a stopping condition has been met. Especially for large models the pending event set may become very large, and a well known issue for good execution time performance is to have an efficient implementation of the set [Jai91, RA97].

A process-oriented model is also based on an underlying event scheduling mechanism but describes the model somewhat differently. A process is associated with each active entity in the model and describes the entity's actions over time. Thus, the flow of control is interrupted and handed over between the different processes by the simulation system so that all actions by different processes are carried out at the appropriate simulated times. It is often claimed that this represents an added level of abstraction over the event-oriented view [WB96], thus reducing the model development time. Event-oriented modelling, on the other hand, can be more execution efficient.

5.1.1 The Activity Scanning Model

The Activity Scan conceptual framework [BL62] is one of the commonly used frameworks in simulations that use fixed time increments (time-steps). The time-flow mechanism [Pag97] adopted by such simulations is shown in Figure 5.1.

To begin with, the model attribute values are assigned their initial values and then the simulation proceeds in a 2-phase loop until the termination condition is satisfied. Phase 1, which is the time-scan phase, is responsible for updating of the simulation clock by a fixed amount. The activity scan phase or Phase 2 forms the core of the simulation wherein each condition is evaluated and if a condition is satisfied then the corresponding actions are performed.

One evaluation of each condition represents a single scan. If any condition is satisfied in a scan then the complete set of conditions must be re-scanned because the actions performed due to the previous condition being satisfied might have caused some new conditions to be satisfied. If no conditions are satisfied in a single scan

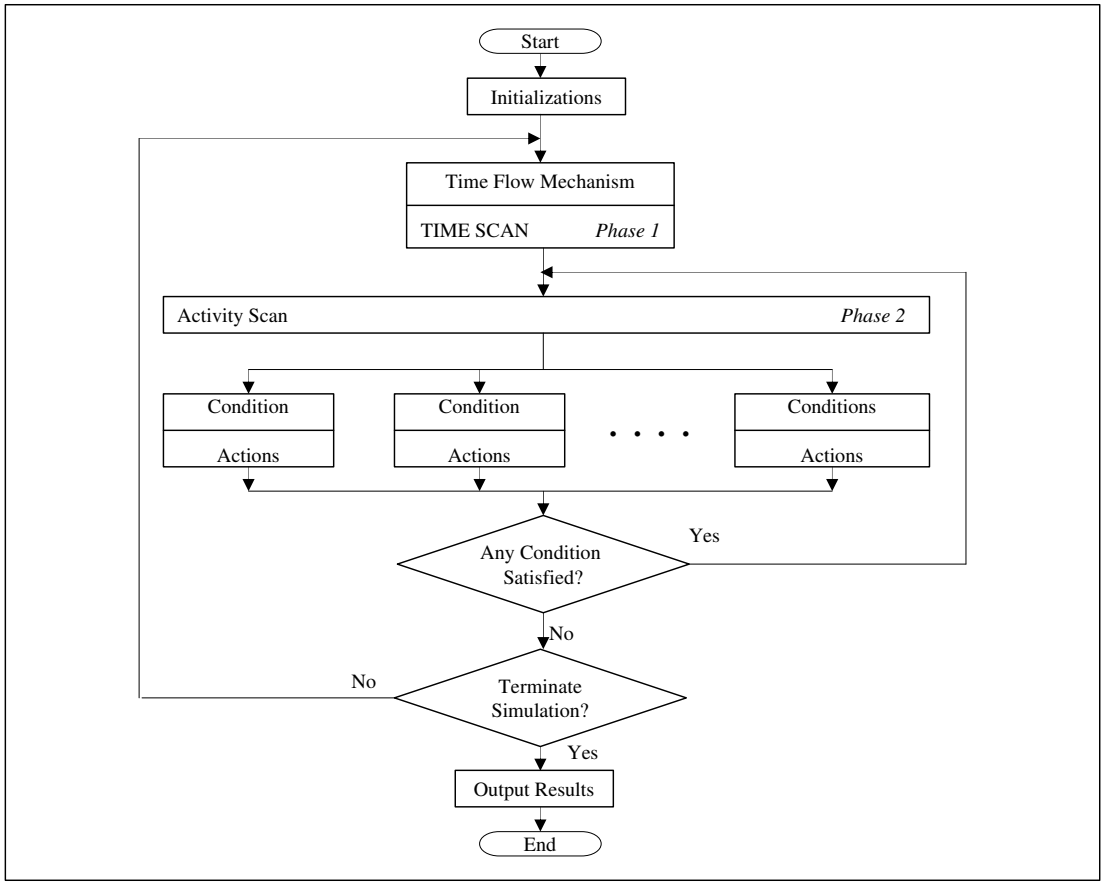

Figure 5.1: Activity Scan Flow Diagram

then Phase 2 terminates.

This technique aims to improve the efficiency of simulations by reducing the synchronization intervals when the event coverage is not dense. Obviously, the most important parameters [Fuj89] that need to be considered are the savings introduced by the new technique in the number of synchronizations, number of overhead messages exchanged and the condition, namely the event coverage, under which these savings (if any) were observed. The parameters to be measured have been chosen based on current literature [Nic90].

The traditional time-stepped mechanism was used as the benchmark. Figure 5.2 shows the number of synchronization points and overhead / time advance messages involved in this technique.

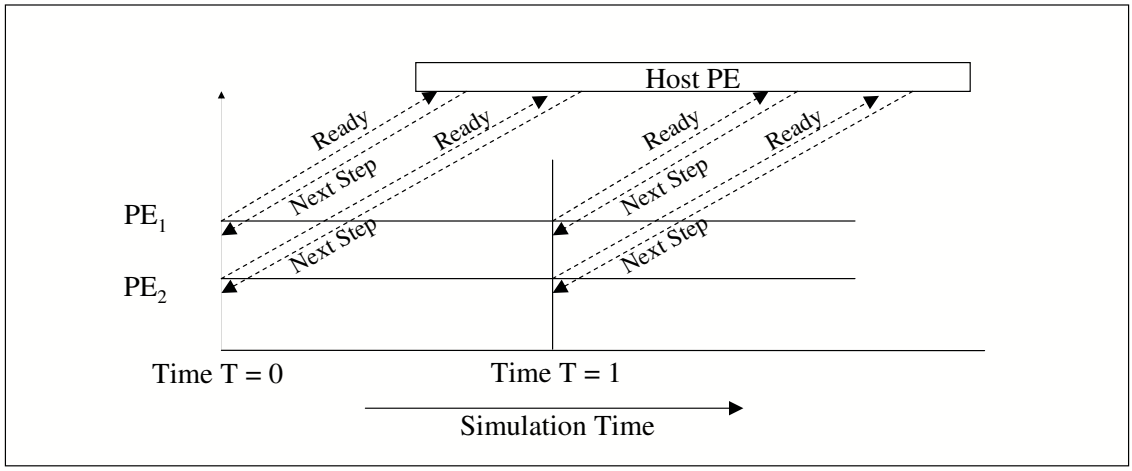

Figure 5.2: Overhead Events and Synchronization Points in Traditional Time-Stepped Technique

It is quite evident from the figure that a minimum of two over-head events are required per entity per unit simulation time for proper synchronization when the traditional technique is used. Since synchronization occurs at every time-step the number of synchronizations are equal to the number of time-steps.

5.1.2 Structure of Each Participating Entity

The important aspects of the test-bed that will be useful to test our techniques are:

- A mechanism to conveniently alter the event coverage of the participating entities
- Easy adaptation to lookaheads
- Provision to introduce new participating entities easily to test scalability

Based on a war-game simulation, we model each entity based on a battle tank. We divide the functionality of a tank into action, physical and behavioral models. The action model calculates the movement co-ordinates of the tank and the physical model implements the actual physical movement. The behavioral models carry out the functionality of initiating patrol missions, patrolling ear-marked territories, identifying potential enemy targets and carrying out attacks. The design of the entities is shown in Figure 5.3.

Events can be generated by any of the component models of the entity and they may be destined to the entity itself or to some other entity. An event received by an entity is handled by identifying the model which is subscribing to an event of that kind and then passing on the event to that model to execute.

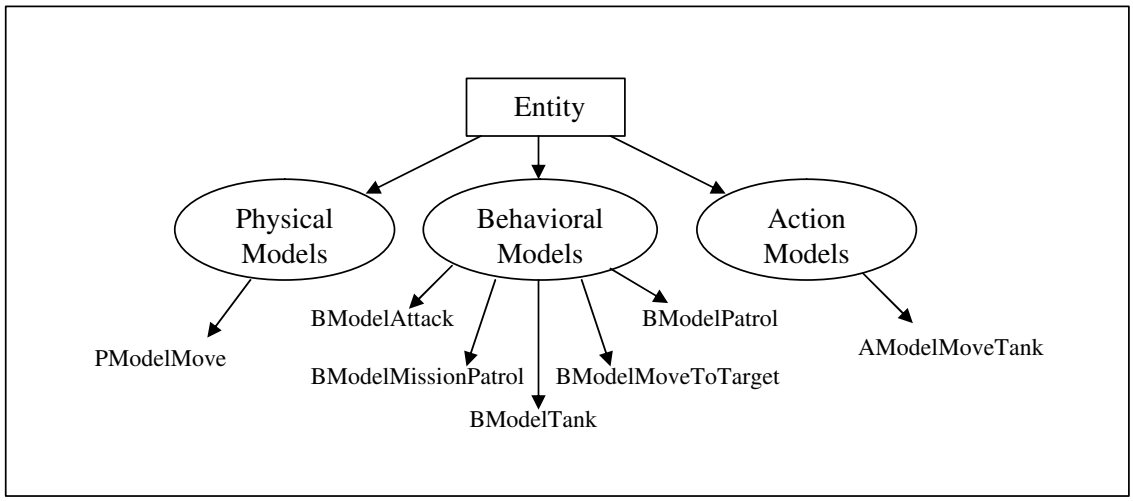

Figure 5.3: Structure of an Entity in the Experimental Framework

5.2 Optimization Using Piggy-backing

MAX_SUPERSTEP_SIZE is a constant that we use to control the maximum super-step size. The coverage of events was controlled randomly by allowing each model in the entity to generate events into the future. Three levels of event coverage were used to study the efficiency of the technique namely 0.25, 0.50 and 0.75.

5.2.1 Results and Observations

The experimental evaluation was done using two different set-ups. The first setup included four PEs and the second setup included eight PEs participating in the simulation on a Local Area Network. The optimized technique was evaluated with respect to the number of overhead messages (as compared to the traditional time-stepped technique) and the physical time overhead introduced at the host due to the introduction of the FTL.

Averages of synchronization messages, overhead messages and time advances across all PEs participating in the simulation were measured using four participating PEs. The experiments were repeated using different values of event coverage. The experiments were also repeated using an eight PE setup and it was observed that the proposed technique is scalable. The observed increase in the number of overhead messages was proportional to the increase in the number of entities participating in the simulation.

The physical time taken to simulate 120000 time units was measured under varying simulation parameters. As the experiments were conducted on a LAN, the savings in time was observed only in cases of very low event coverage (for piggybacking based simulation) and in cases of high lookahead (for lookahead based simulation). The overhead events as a percentage of the total events simulated has also been presented.

5.2.1.1 Response to Changes in the Super-step Throttle

The Super-step throttle dictates the size of the super-step used by the technique to reduce the time-step synchronization costs. From Figures 5.4 and 5.5 we can see that larger values of χ can lead to an increase in the size of some super-steps and thus potentially cause savings in the number of global time-step synchronizations.

However, large values of χ also means wider periods between consecutive super-steps which is not necessarily advantageous. As discussed before, this technique aims to speed-up fine-grained time-stepped simulations that have high event coverage in some regions and low in others. Hence large super-steps preclude that the event coverage will not change within the span of the super-step which can be dangerous.

Consider the following example: At time T=200, we decide to use a super-step of size 50 and at T=210 we have a declaration for war. It is now not possible to switch to traditional time-stepped mode, which is more efficient in the prevailing circumstances, until T= 250.

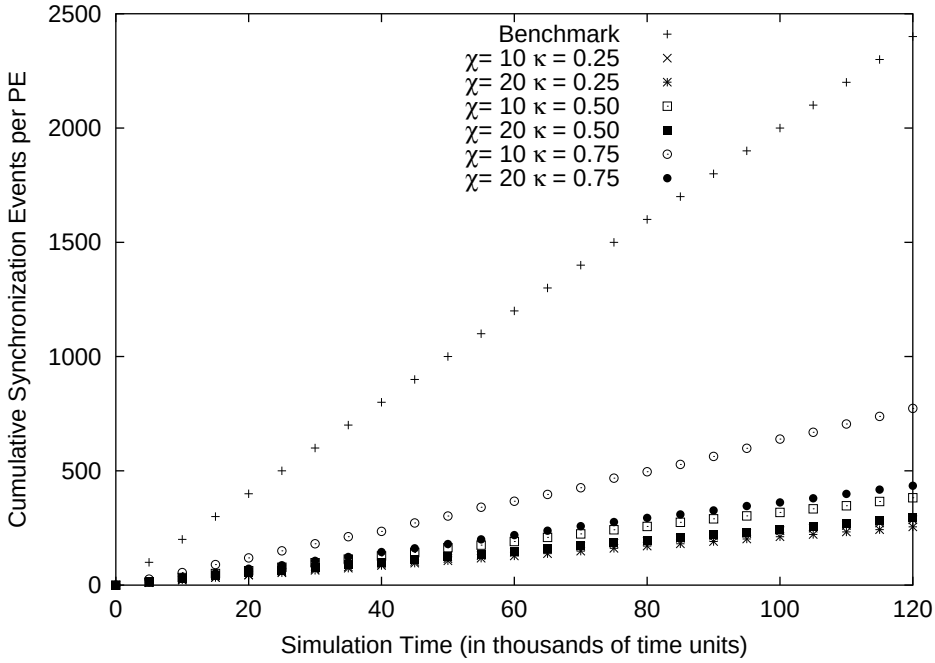

Figure 5.4: Cumulative Global Synchronization Events (4 PEs)

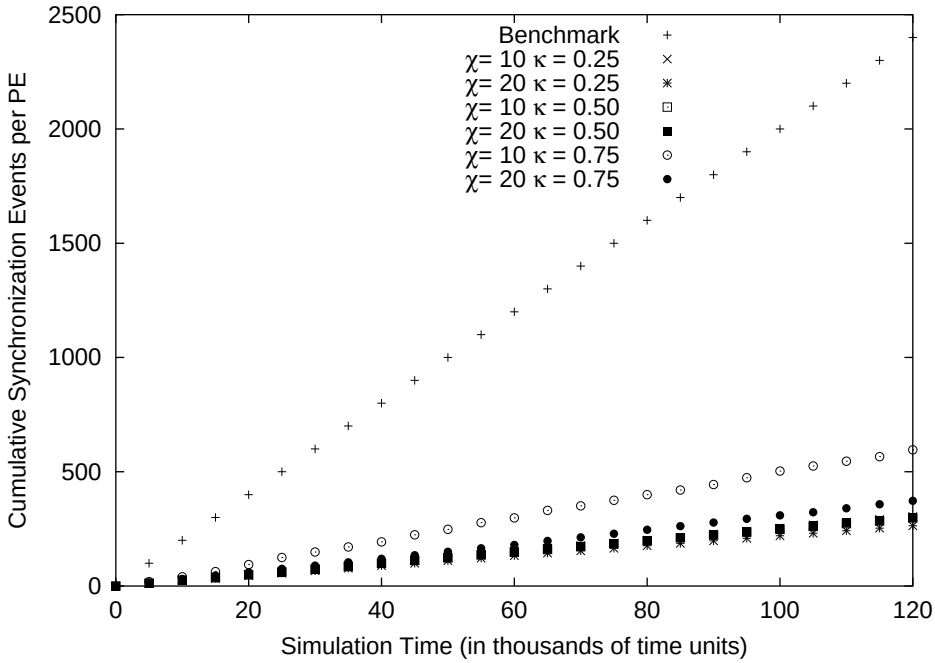

Figure 5.5: Cumulative Global Synchronization Events (8 PEs)

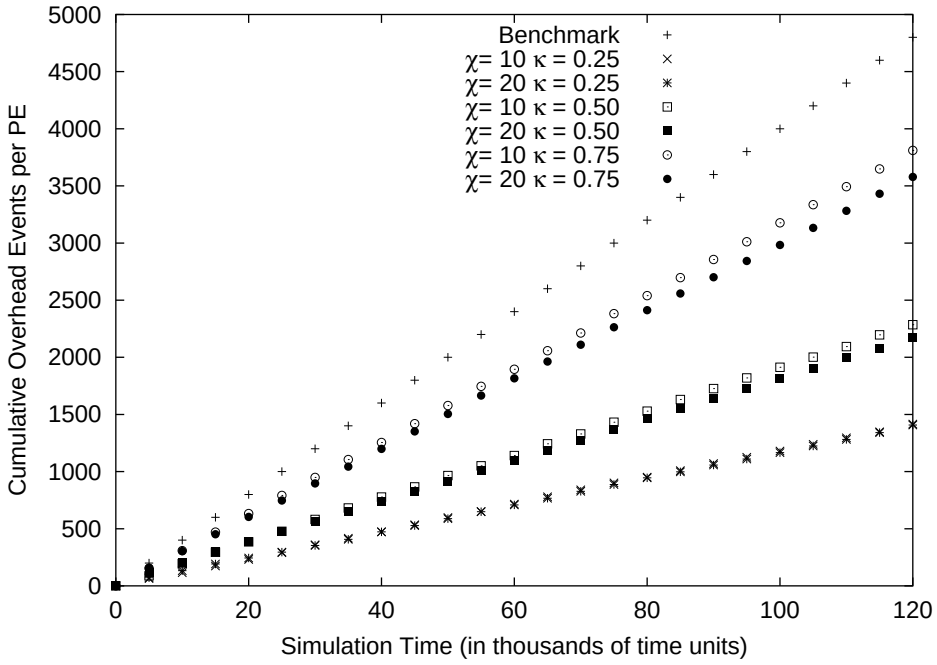

Figure 5.6: Cumulative Overhead Events (4 PEs)

5.2.1.2 Response to Event Coverage

From the graphs for overhead events (Figure 5.6 and Figure 5.7) it is clear that the number of synchronizations and the number of overhead messages are less than the benchmark in all cases. This can also be seen from the analysis shown in the previous section.

Recall that the worst-case overhead based on linear regression from the previous section was shown to be 2 events per time unit versus 2 events per time unit of the benchmark algorithm. Hence, when the event coverage equals 1, the performance of our proposed technique equals that of the conventional technique in terms of overhead messages. This is evident from the experimental values obtained. The number of overhead messages, which is a crucial consideration factor in distributed simulations running on a LAN/WAN, is proportional to the event coverage in the proposed technique.

The introduction of super-steps makes it convenient to switch to traditional time-stepping whenever the event-coverage increases beyond a threshold value. From the graphs for time advance events (Figure 5.8 and Figure 5.9) we can observe that the

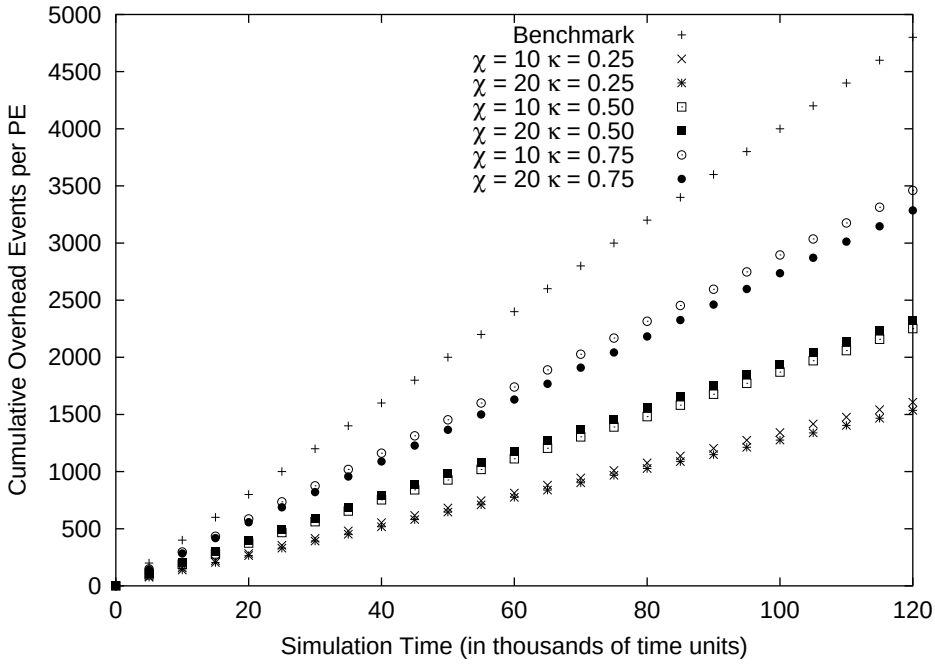

Figure 5.7: Cumulative Overhead Events (8 PEs)

number of time advances sent out by the Host increases with an increase in *Event Coverage* when the piggy-backing technique is used.

To maintain high simulation efficiency we can switch to traditional time-stepping when the *Event Coverage* crosses a threshold (for instance, 0.80) in order to avoid the calculation overheads at each time-step that the piggy-backing technique entails.

An important property of the piggy-backing technique is that the super-step size can be pegged to 1 at any point of the simulation that will automatically cause the simulation to behave in the traditional time-stepped mode. This is useful in cases where we would like to have the advantage of fine-granularity without having to worry about availability of events at each time unit.

5.2.1.3 Overhead Time

The introduction of the FTL results in some overhead at the host. Since we are reducing the number of overhead messages at the cost of introducing an additional data structure at the central host it is important to measure the physical time spent by the host in maintaining the data structure. This time is affected by two

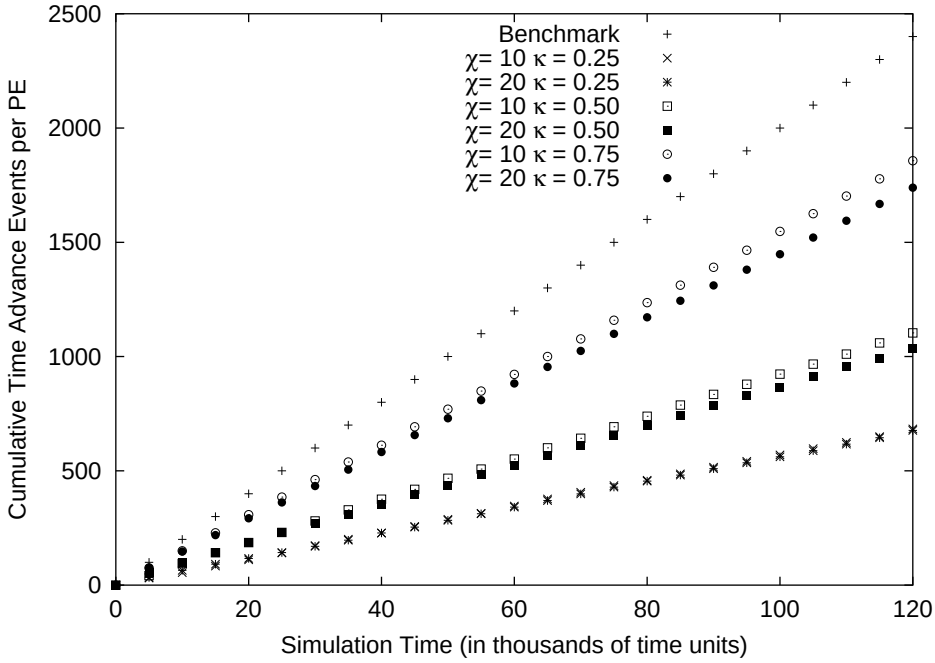

Figure 5.8: Cumulative Time Advance Events (4 PEs)

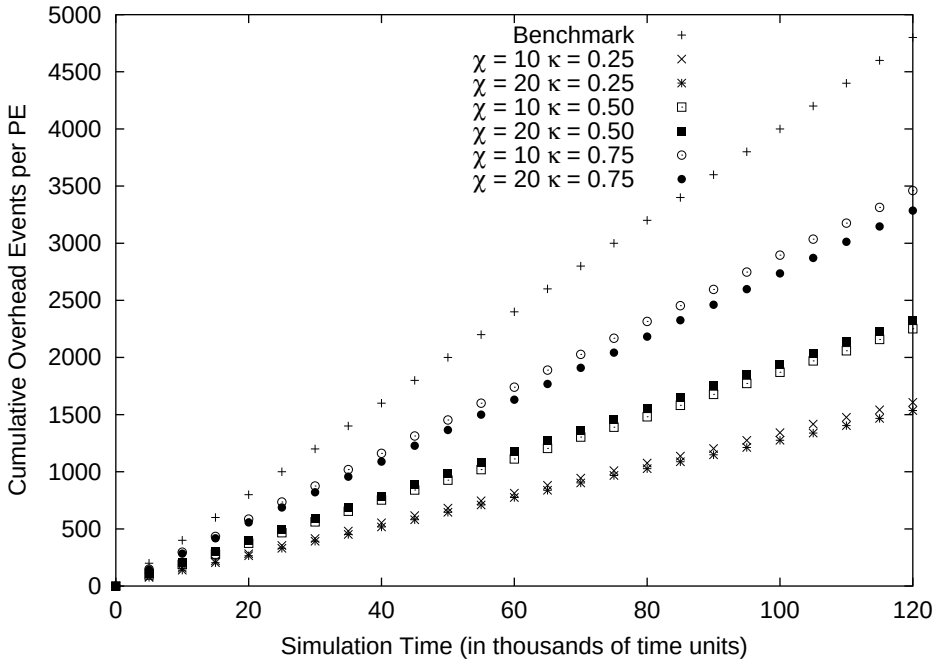

Figure 5.9: Cumulative Time Advance Events (8 PEs)

parameters: the number of time-stamps already in the FTL and the number of time-stamps newly introduced. We varied these two parameters from 0 to 500 and observed (Figure 5.10) that when both the parameters were varied beyond 350, the physical time taken ranged between 0.005 to 0.020 seconds.

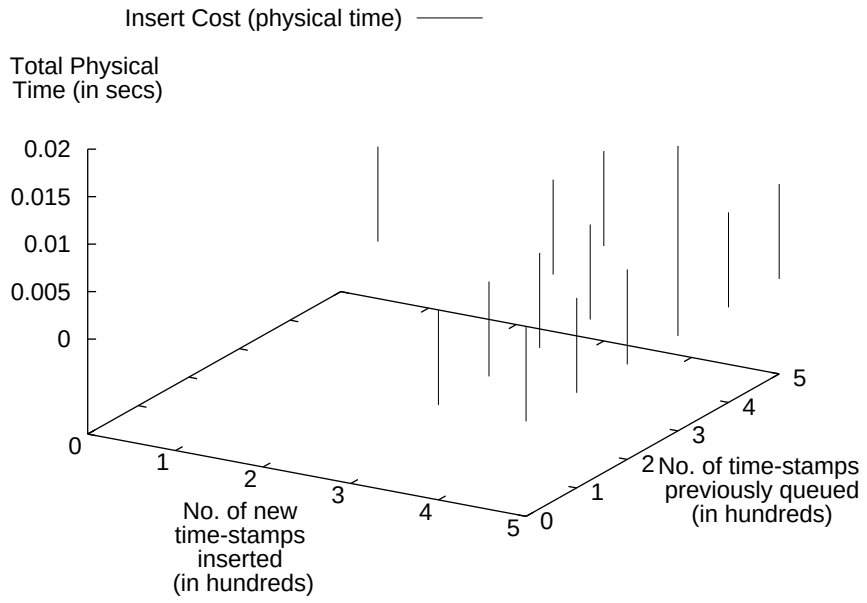

Figure 5.10: Overhead Physical Time at Central Host

The overhead time consumed for piggy-backing information on to the *Ready* messages was also measured. However, the time consumed was negligible ($\leq$ 10 msec) even when 1000 new time-stamps were piggy-backed. The observed results are shown in Figure 5.11.

5.3 Optimization Using Lookaheads

Again, MAX_SUPERSTEP_SIZE was used to control the maximum super-step size. The coverage of events was controlled randomly by allowing each model in the entity to generate events into the future. Three levels of event coverage were used to study the efficiency of the technique namely 0.25, 0.50 and 0.75.

The experimental evaluation was done using a four PE setup and the lookaheads were varied from 1 to 3.

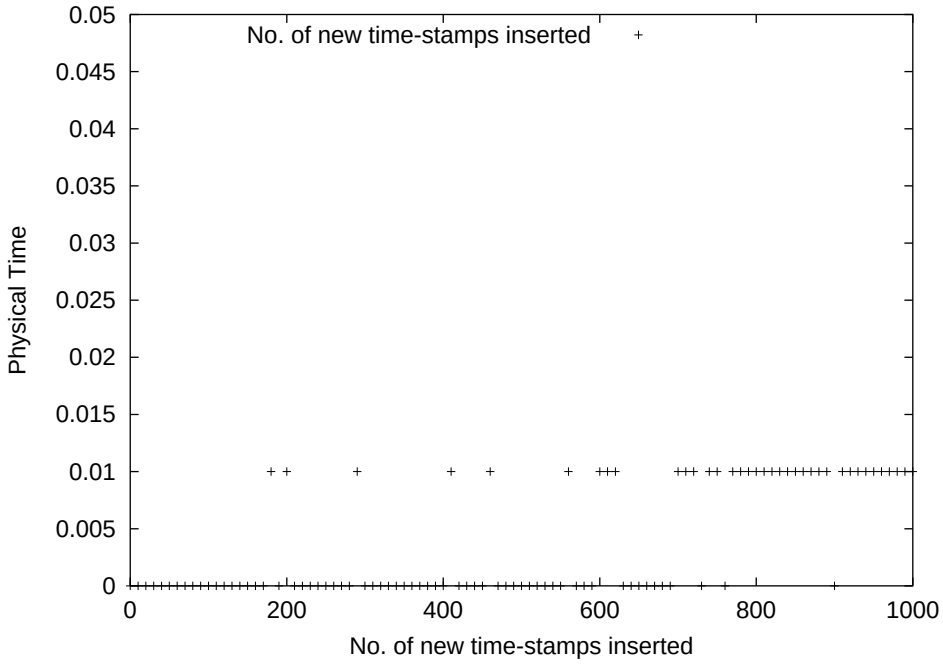

Figure 5.11: Overhead Physical Time at PEs

5.3.1 Results and Observations

From Figure 5.12 it is evident that the number of synchronizations reduce with an increase in lookahead. This has been discussed in detail in [Fuj89] which gives a clear insight about lookaheads and its affect on distributed simulations.

5.3.1.1 Response to Lookahead

Figure 5.13 shows that as average lookahead values decrease the number of overhead events increase. If the number of entities participating in the simulation are large and the average lookahead values between entities is low then the cost of achieving a barrier is substantially higher than the cost of using the the traditional time-stepped technique.

On a general note, a barrier is more efficient than the optimized time-stepped technique (using piggy-backing) when the product of the average event coverage across the barrier (denoted by κ_{Bar}) and the barrier length itself (denoted by $|Barrier|$) is greater than 1, that is, $\kappa_{Bar} * |Barrier| > 1$. Similarly, a barrier is more efficient than the traditional technique when $|Barrier| > 1$. This leads us to the need for change in simulation mode depending on changes in event coverage and density.

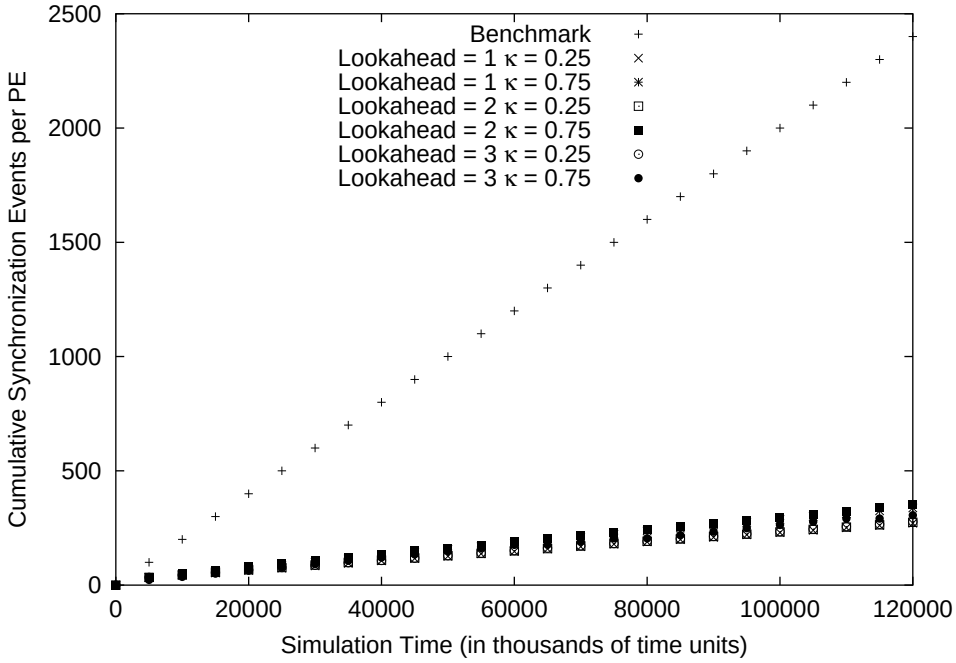

Figure 5.12: Cumulative Global Synchronization Events (4 PEs with Lookahead)

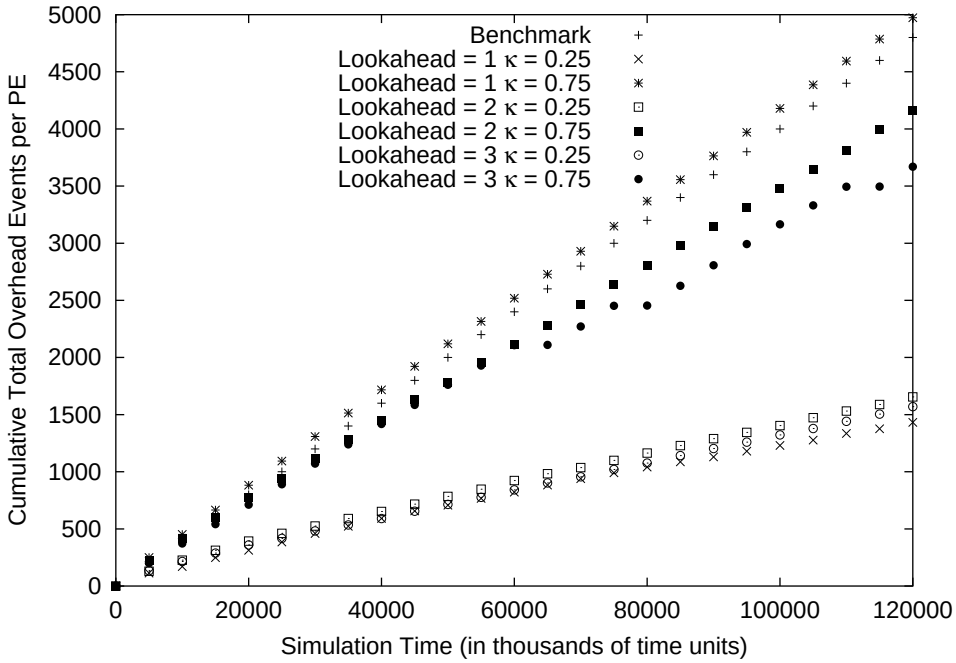

Figure 5.13: Cumulative Overhead Events (4 PEs with Lookahead)

Figure 5.14 plots the average number of time advances across all PEs participating in the simulation. The number of time advances also show a decrease with an increase in the average lookaheads of the participating PEs.

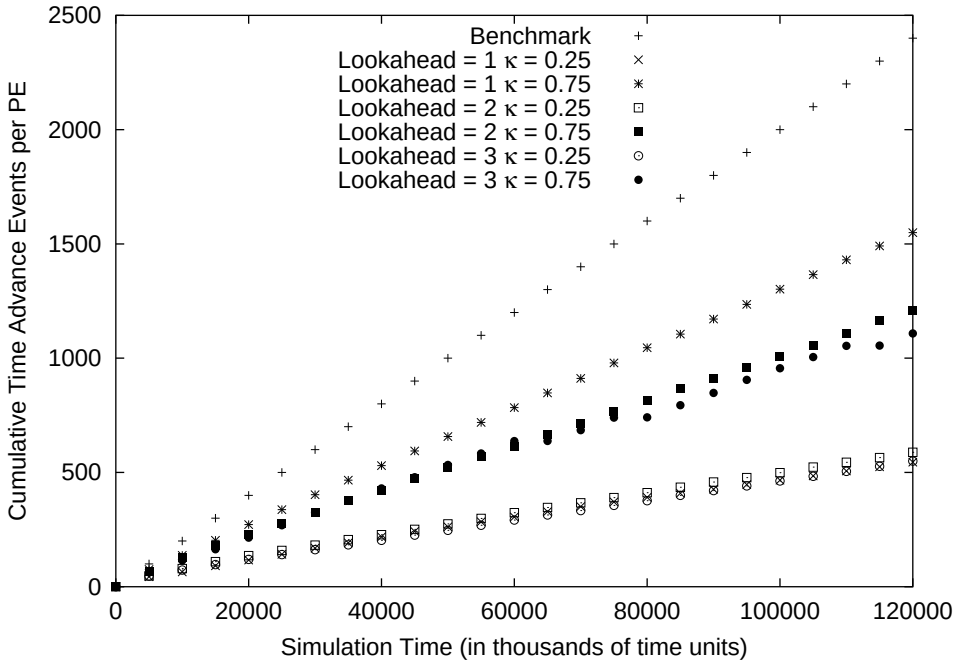

Figure 5.14: Cumulative Time Advance Events (4 PEs with Lookahead)

5.3.1.2 Need for Change in Simulation Mode

From the experimental results shown in Sections 5.2.1 and Figures 5.3.1, we can conclude that the efficiency of a simulation technique depends on the parameters of the PEs and their inter-communication cost. Hence, there is a high possibility of increasing efficiency by switching simulation techniques from time to time depending on prevailing conditions.

By dividing the simulation into super-steps we develop a technique where we can monitor the state of the simulation at each super-step boundary and take a decision about the technique to be used in the next super-step. The decision is based entirely on the information gathered from the participating PEs and is as scalable as the underlying barrier or time-stepping technique used because all the information is piggy-backed on the overhead messages used for synchronization purposes.

5.4 Physical Simulation Time and the Overhead Ratio

The physical time taken for a simulation using 4 PEs to simulate 120000 time units was measured using different values of the event coverage κ, in case of optimization using piggybacking and different values of lookahead in case of the lookahead based technique.

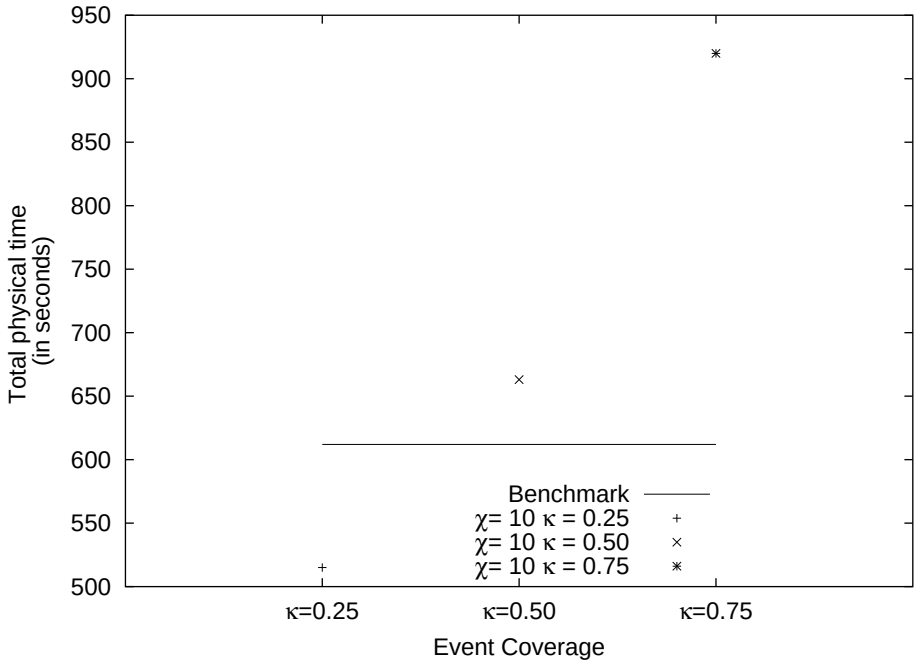

Figure 5.15: Physical Total Time for Simulation under varying Simulation Parameters (for 120000 time units, 4 PEs) using Piggybacking

Figure 5.15 shows the total time taken when the simulation using the Piggybacking technique was repeated using different values of Event Coverage. The participating PEs were all residing on the same LAN. As the inter-PE communication costs in case of LANs is relatively low compared to MANs and WANs, we can see that the optimizations are beneficial only in a minority of cases. In our experiments only simulations with low event coverage ($\kappa = 0.25$) yielded better results than the benchmark. For $\kappa = 0.50$, the total physical time taken for the simulation was slightly greater than that for the benchmark. This experimental result is consistent with the analysis of the optimization technique because we have reduced the inter-PE

communication messages by introducing extra processing overheads mainly at the Host. When communication costs are low, as in a high-speed LAN environment, the benefits of this optimization can be seen only when event coverage is low enough to outweigh the extra processing overhead. If the distributed simulation is residing on a WAN, the performance improvement will be more noticeable and the technique will out-perform conventional time-stepped simulation for higher values of event coverage.

In the case of Lookahead based simulation, with higher values of lookahead the physical time taken for the simulation progressively reduces as shown in Figure 5.16

.

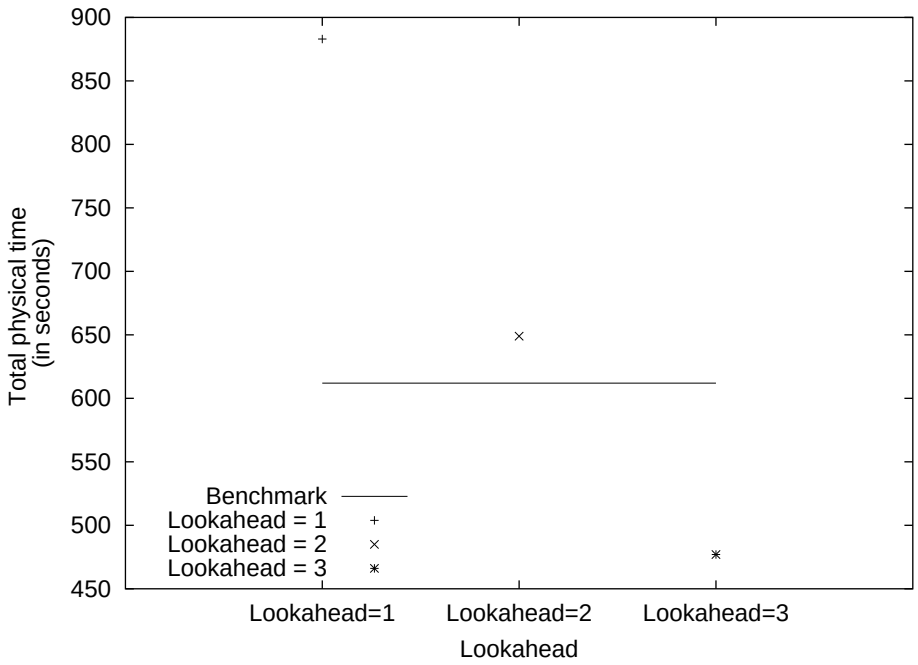

Figure 5.16: Physical Total Time for Simulation under varying Simulation Parameters (for 120000 time units, 4 PEs) using Lookahead

The number of overhead events encountered in the piggybacking based simulation as a percentage of the total events simulated is shown in Figure 5.17. The benchmark which uses conventional time-stepped technique encounters the same number of overhead events in all cases. The overhead encountered by the optimized technique depends on factors such as how sparse the event set is and how uniformly distributed the events are. Measurements indicate an increase in the ratio of overhead events

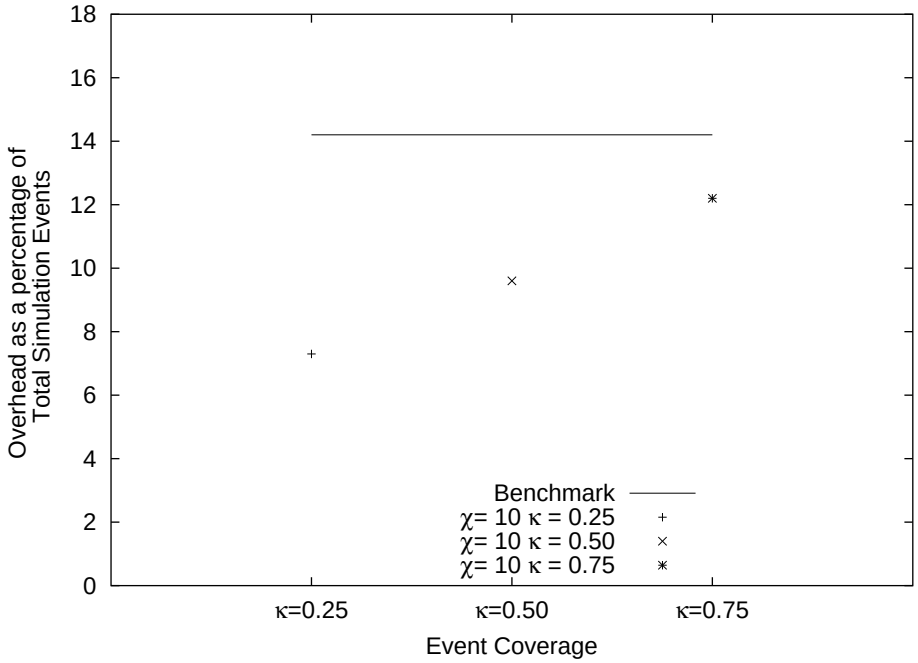

Figure 5.17: Overhead Events as a percentage of Total Simulation Events under varying Simulation Parameters for Piggybacking Based Simulation

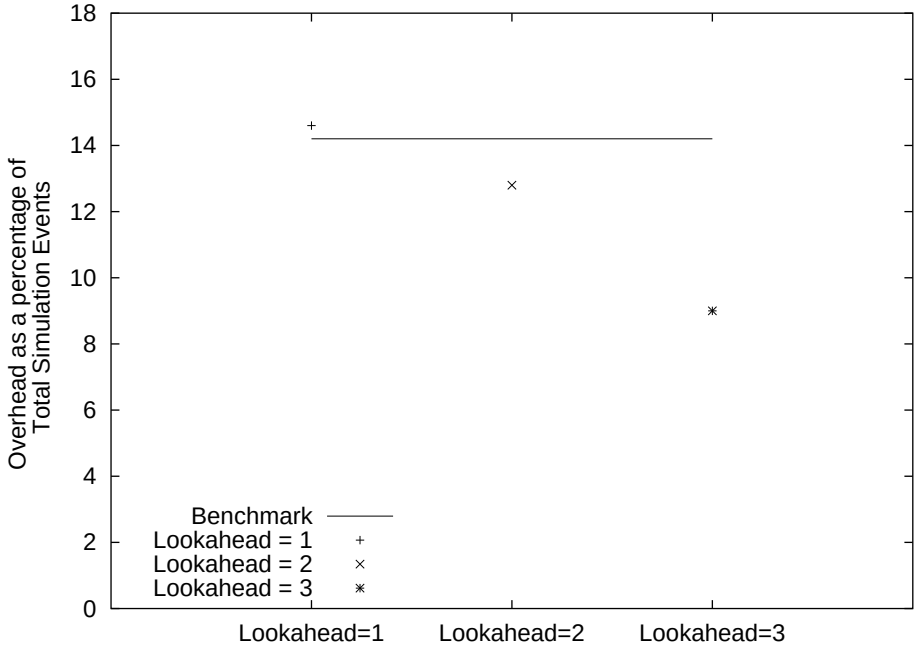

Figure 5.18: Overhead Events as a percentage of Total Simulation Events under varying Simulation Parameters for Lookahead Based Simulation

to total events with increase in event coverage. Thus for simulations with very high event coverage it can be beneficial to use the benchmark technique.

In case of lookahead based simulation, the overhead ratio fell with increase in lookahead as shown in Figure 5.18. This is because larger values of lookahead imply larger time spans between global synchronizations leading to lower overhead costs.

Chapter 6

Conclusions and Further Work

Parallel and distributed simulations offer substantial performance benefits over sequential simulations when the overheads incurred are properly managed. The time-stepped paradigm is often used for implementing simulations that have to adhere to strict real-time constraints. Though the synchronization technique in the case of time-stepped simulations is a simple and robust technique, its performance is heavily dependent on the density and the spread of events. Further, when tight real-time constraints can be relaxed, available lookaheads can be used to improve the performance of the simulation.

6.1 Summary

We have proposed two optimizations - piggy-backing based and lookahead based, to the traditional time-stepped simulation. The optimization by piggy-backing is achieved by maintaining information about future events at the host. We implemented this by piggy-backing the *Ready* messages with information about the new events spawned in the previous time-step. The Host maintained a list which kept track of the PEs that needed to be informed for each future time-step. This list was updated based on the future event information that was piggy-backed on the *Ready* signals received from the PEs. PEs were involved in the synchronization only when they had events to be processed and hence the technique obviated the need for global synchronization at each time-step.

The introduction of the FTL at the Host resulted in increased processing overhead at the Host at each time-step as compared to the traditional time-stepping technique. The extra processing overhead involved was measured and was found to be negligible (< 10 msec) for most practical cases.

The technique was well-suited for interactive simulations as the global simulation state was available at regular and frequent points in time. Further, tight real-time constraints could be obeyed as any event bearing a particular time-stamp t, was processed before $t + \Delta$.

In cases where lookahead information was available and tight real-time constraints were relaxed a barrier synchronization based simulation was used. At each barrier, the Host calculated a lower bound on the time-stamps of receivable events for each PE and informed this time to the PE. The PE then processed all events between its current simulation time and the time declared as safe by the Host. The complexities in maintaining strict real-time constraints in this method were identified.

During the progress of a simulation, it is possible that PE parameters such as event density, lookahead and simulation parameters such as real-time constraints vary from time to time. We maximized simulation efficiency by switching between techniques that were suited to the prevailing environment. To achieve this switching we introduced the 'super-stepping' technique which enabled us to switch between traditional, piggyback-based and lookahead-based simulation techniques. A probabilistic method to estimate suitable 'super-step' sizes based on observed event history was also introduced. The technique involved fitting a Poisson distribution to the incoming events and then predicting the future density and coverage of events based on it. In cases where a Poisson fit failed, a simple regression fit was used. These super-steps served two main purposes, *viz.* easy switching between different techniques of simulation and convenient global state saving and strategy manipulation.

Our experimental results showed that substantial savings in overhead can be achieved when these optimizations were used in simulations, when compared with

the conventional time-step technique. The proposed techniques were simple and robust. The savings in overhead messages were achieved at the cost of introducing some processing overhead at the Host in terms of updating its FTL at each time-step. The physical time taken by the Host to perform this update was also measured.

6.2 Further Work

In this section we highlight two unresolved issues related to the work presented in this thesis.

6.2.1 Clustering in Case of Non-Total Dependency

The techniques discussed so far assumed that there is total-dependency between the participating entities of the simulation. However, there exists the possibility that the entities can be divided into sub-sets such that each sub-set contains entities with zero-lookahead amongst each other but some non-zero positive lookahead exists between any member entity of one set and any member entity of another.

Consider a war-gaming situation where infantry, air support and naval support are being simulated in a typical war scenario. If entities representing land units, air units and naval units are clustered separately and if we can assign lookaheads between these clusters then we can detect safe zones in which the clusters can proceed with the simulation in parallel.

Figure 6.1 depicts the advancing of simulation. At T = 2, the simulation executive scans through the lookahead table and determines the next 2 time units to be inter-cluster event safe. Hence the activity scans of the 2 clusters can proceed in parallel for the next 2 time units. After the completion of the 2 time units the inter-cluster lookaheads might have changed leading to the determination of 4 time units to be inter-cluster event safe by the simulation executive and so on.

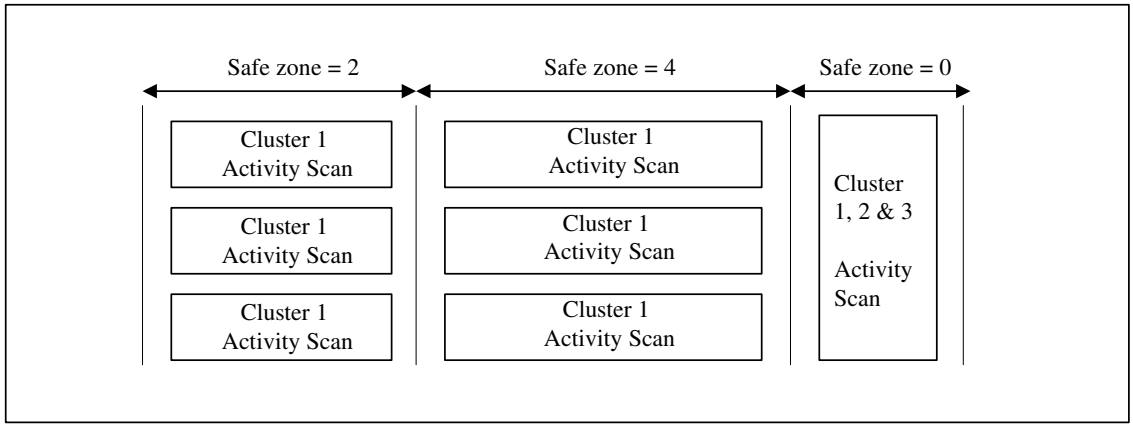

Figure 6.1: Progress of an Inter-Cluster Lookahead Based Simulation

Between two consecutive inter-cluster synchronizations the clusters can adapt independent simulation paradigms based on prevailing simulation parameters.

6.2.2 Communication Cost versus Processing Overhead

Simulation techniques encounter a degraded performance when the communication overhead or the processing overhead outweighs the computation granularity. Depending on the communication backbone employed by the simulation setup it is important to identify the constraining parameter among the two. It is not worthwhile to introduce heavy processing overhead in order to reduce communication cost between PEs if the underlying communication system is extremely fast as in the case of shared memory systems. In this aspect further work is necessary to determine the break-even point where the additional processing overhead introduced by a technique justifies the savings in communication cost.

Bibliography

[AR94] R. Ayani and H. Rajaei. Parallel simulation based on conservative time windows: a performance study. In *Concurrency: Practise and Experience*, volume 6(2), pages 119–142, 1994.

[Aya89] R. Ayani. A parallel simulation scheme based on distance between objects. In *Proceedings of the SCS Multiconference on Distributed Simulations*, 1989.

[BD97] A. Boukerche and S.K. Das. Distributed interactive and real-time simulations. Technical Report O-8186-7773-2/97, IEEE, 1997.

[BL62] J.N. Buxton and J.G. Laski. Control and simulation language. In *Computer Journal*, volume 5(3), 1962.

[BS88] W.L. Bain and D.S. Scott. An algorithm for time synchronization in distributed discrete event simulation. In *Distributed Simulation*, volume 19(3) of *SCS Simulation Series*, pages 30–33. Society for Computer Simulation, 1988.

[CLT97] W. Cai, E. Letertre, and S.J. Turner. Dag consistent parallel simulation: a predictable and robust conservative algorithm. In *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*, pages 178–181, 1997.

[CM79] K.M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. In *IEEE Transactions on Software Engineering*, volume 5(5), pages 440–452, 1979.

[CM81] K.M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. In *Communications of the ACM*, volume 24(11), pages 198–205, 1981.

[CS89] K.M. Chandy and R. Sherman. The conditional event approach to distributed simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 93–99, 1989.

[CT90] W. Cai and S.J. Turner. An algorithm for distributed discrete-event simulation ?the carrier null message approach. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 3–8. Society for Computer Simulation, 1990.

[DJ91] P.M. Dickens and P.F. Reynolds Jr. A performance model for parallel simulation. In *Proceedings of the Winter Simulation Conference*, 1991.

[EGLW93] S.G. Eick, A.G. Greenberg, B.D. Lubachevsky, and A. Weiss. Synchronous relaxation for parallel simulations with applications to circuit-switched networks. In *ACM Transactions on Modeling and Computer Simulation*, volume 3(4), pages 287–314, 1993.

[Eks00] U. Ekstrom. Design patterns for simulations in Erlang/OTP. Master's thesis, Uppsala University, 2000.

[Fer95] A. Ferscha. Parallel and distributed simulation of discrete event systems. In *Handbook of Parallel and Distributed Computing*. McGraw Hill, 1995.

[FK87] A.S. Fotheringham and D. C. Knudsen. *Goodness-of-Fit Statistics*. Norwich : Geo Books, 1987.

[FN92] R. Fujimoto and D.M. Nicol. State of the art in parallel simulation. In *Proceedings of the Winter Simulation Conference*, 1992.

[Fuj89] R.M. Fujimoto. Performance measurements of distributed simulation strategies. In *Transactions of the Society for Computer Simulation*, volume 6, pages 89–132, 1989.

[Fuj90] R.M. Fujimoto. Parallel discrete event simulation. In *Communications of the ACM*, volume 33(10), pages 30–53, 1990.

[Fuj93] R. Fujimoto. Parallel and distributed discrete event simulation: Algorithms and applications. In *Proceedings of the Winter Simulation Conference*, 1993.

[Fuj97] R.M. Fujimoto. Zero lookahead and repeatability in the high level architecture. In *Proceedings of the Spring Simulation Interoperability Workshop*, 1997.

[Fuj00] R.M. Fujimoto. *Parallel and Distributed Simulation Systems.* Wiley Interscience, 2000.

[GLC+01] B.P. Gan, Y.H. Low, W. Cai, S.J. Turner, S. Jain, W.J. Hsu, and S.Y. Huang. The development of conservative superstep protocols for shared memory multiprocessor systems. In *Parallel and Distributed Computing Practices*, volume 4(1), pages 1–17, 2001.

[GT88] B. Groselj and C. Tropper. The time-of-next-event algorithm. In *Distributed Simulation*, volume 19(3) of *SCS Simulation Series*, pages 25–29. Society for Computer Simulation, 1988.

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley-Inter-science, 1991.

[JBN96] J.S. Carson J. Banks and B.L. Nelson. *Discrete-Event System Simulation.* Prentice Hall, second edition, 1996.

[Jef85] D.R. Jefferson. Virtual time. In *ACM Transactions on Programming Languages and Systems*, volume 7(3), pages 404–425, 1985.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. In *Communications of the ACM*, volume 21(7), 1978.

[LK91] A. Law and W. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, second edition, 1991.

[Lub88] B.D. Lubachevsky. Bounded lag distributed discrete event simulation. In *Distributed Simulation*, volume 19(3) of *SCS Simulation Series*, pages 183–191. Society for Computer Simulation, 1988.

[Lub89a] B.D. Lubachevsky. Efficient distributed event-driven simulations of multiple-loop networks. In *Communications of the ACM*, volume 32(1), pages 111–123, 1989.

[Lub89b] B.D. Lubachevsky. Scalability of the bounded lag distributed discrete event simulation distributed simulation. In *Distributed Simulation*, volume 21(2) of *SCS Simulation Series*, pages 100–107. Society for Computer Simulation, 1989.

[Man97] P.S. Mann. *Introductory Statistics*. John Wiley and Sons, third edition, 1997.

[Mis86] J. Misra. Distributed discrete-event simulation. In *ACM Computing Surveys*, volume 18(1), 1986.

[Nic88] D.M. Nicol. Parallel discrete-event simulation of FCFS stochastic queueing networks. In *SIGPLAN Notices*, volume 23(9), 1988.

[Nic90] D.M. Nicol. The cost of conservative synchronization in parallel discrete event simulations. In *Journal of the ACM (JACM)*, volume 40(2), pages 304–333, 1990.

[Nic91] D.M. Nicol. Performance bounds on self-initiating discrete-event simulations. In *ACM Transactions on Modelling and Computer Simulation*, volume 1(1), pages 24–50, 1991.

[Pag97] E.H. Page. Zero lookahead in a distributed time-stepped simulation. In *Simulation Digest*, volume 26(2), pages 4–13, 1997.

[RA97] R. Ronngren and R. Ayani. A comparative study of sequential and parallel priority queue algorithms. In *ACM Transactions on Modeling and Computer Simulation*, volume 7(2), pages 157–209, 1997.

[SBW88] L.M. Sokol, D.P. Briscoe, and A.P. Wieland. MTW: A strategy for scheduling discrete simulation events for concurrent execution. In *Distributed Simulation*, volume 19(3) of *SCS Simulation Series*, pages 34–42. Society for Computer Simulation, 1988.

[Tay98] S.C. Tay. *Parallel Simulation Algorithms and Performance Analysis*. PhD thesis, National University of Singapore, 1998.

[THFD98] K. Townsend, Z. Haraszti, J. Freebersyser, and M. Devetsikiotis. Simulation of rare events in communications networks. *IEEE Communications Magazine*, 1998.

[TS02] G.S.H. Tan and K. Shenoy. Cluster-based distributed simulation using HLA. Technical report, NUS and CSO, MINDEF, 2002.

[TTS03] S.C. Tay, G.S.H. Tan, and K. Shenoy. Piggy-backed time-stepped simulation with 'super-stepping'. In *Proceedings of the Winter Simulation Conference*, 2003.

[WB96] P. Wonnacott and D. Bruce. The APOSTLE simulation language; granularity control and performance data. In *Proceedings of the 10th Workshop on Parallel and Distributed Simlation*, 1996.

[WT94] K.R. Wood and S.J. Turner. A generalized carrier-null method for conservative parallel simulation. In *Proceedings of SCS Parallel and Distributed Simulation Conference*, pages 50–57, 1994.